Developing Cyberspace Data Understanding: Using CRISP-DM for Host-based IDS Feature Mining

THESIS

Joseph R. Erskine, Captain, USAF

AFIT/GCS/ENG/10-01

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/10-01

Developing Cyberspace Data Understanding: Using CRISP-DM for Host-based IDS Feature Mining

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

Joseph R. Erskine, B.S.C.S.

Captain, USAF

March 2010

AFIT/GCS/ENG/10-01

Developing Cyberspace Data Understanding: Using CRISP-DM for Host-based IDS Feature Mining

Joseph R. Erskine, B.S.C.S.
Captain, USAF

Approved:

/signed/	12 Mar 2010
Dr. Gilbert L. Peterson (Chairman)	date
/signed/	12 Mar 2010
Dr. Barry E. Mullins (Member)	date
/signed/	12 Mar 2010
Dr. Michael R. Grimaila (Member)	date

AFIT/GCS/ENG/10-01

Abstract

Current intrusion detection systems (IDS) generate a large number of specific alerts, but do not provide actionable information. Many times, these alerts must be analyzed by a network defender, a time consuming and tedious task which can occur hours or days after an attack. Improved understanding of the cyberspace domain can lead to great advancements in cyberspace situational awareness research and development. This thesis applies the Cross Industry Standard Process for Data Mining (CRISP-DM) to develop an understanding about a host system under attack. Data is generated by launching scans and exploits at a machine outfitted with a set of host-based data collectors. Through knowledge discovery, features are identified within the data collected which can be used to enhance host-based intrusion detection. By discovering relationships between the data collected and controlled events, false positive alerts were reduced by over 91% when compared to a leading open source IDS. This method of searching for hidden forensic evidence relationships enhances understanding of novel attacks and vulnerabilities, bolstering ones ability to defend the cyberspace domain. The methodology presented in this thesis, as well as the features identified, can be used to further situational awareness research.

Dedicated to Grammy and Grampy, who have filled my childhood and beyond with warm and happy memories.

Acknowledgements

First and foremost, I owe a large debt of gratitude to God. It is through His good graces alone that I succeed.

To my advisor and committee, thank you for your instruction and mentorship. Because you have challenged me to do more than I thought possible, I have learned so much through this process.

To my wife and children, thank you for your patience, love and support throughout this process. You have been my rock and my inspiration.

Mom and Dad, thank you for guiding me to stay on the right path, encouraging me to give my best, and for always believing in me.

Last but not least, I'd like to thank those from my career (you know who you are) who have inspired me to grow, from my days as an Airman Basic through today, still an Airman. I stand upon your shoulders.

Joseph R. Erskine

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation		Page
DEU	Data Extraction Utility	35
SAIL	Situational Awareness Inference and Logic	36
CNL	Controlled natural language	36
INFERD	INformation Fusion Engine for Real-time Decision-making	38
KDD	Knowledge Discovery and Data mining	39
ACM	Association for Computing Machinery	39
SIGKDD	Special Interest Group for KDD	39
DMG	Data Mining Group	39
CRISP-DM	CRoss Industry Standard Process for Data Mining	39
WEKA	Waikato Environment for Knowledge Analysis	41
ANN	Artificial neural network	42
ROC	Receiver operating characteristic	44
MIT	Massachusetts Institute of Technology	48
DARPA	Defense Advanced Research Projects Agency	48
WSH	Windows Script Host	54
VM	Virtual Memory	56
GUI	Graphical User Interface	57
DLL	Dynamic-linked library	58
ETW	Event Trace for Windows	61
ETL	Event trace log	61
PID	Process ID	68
SPID	Standard Process ID	68
FTW	Forensic Timestamp Window	69
MANET	Mobile Ad hoc Network	69
T-SQL	Transact SQL	70
UNC	Universal Naming Convention	75
MUP	Multiple UNC Provider	75
API	Application Program Interface	99

Developing Cyberspace Data Understanding: Using CRISP-DM for Host-based IDS Feature Mining

I. Introduction

Cyberspace situational awareness research is an effort to build and refine human understanding of the cyberspace domain, the globally interconnected set of computing and communication resources upon which the Department of Defense (DoD) relies upon for carrying out its mission. There are a number of motivations for enhancing cyberspace situational awareness. Among these are the DoD's increased reliance upon the cyberspace domain, the continually expanding arsenal of cyberspace threats, and the tradeoffs between cyberspace threat detection effectiveness versus efficiency and human understanding. The focus of this document is on developing understanding, specifically related to host-based intrusion detection, and to determine if such a methodology could be used to focus the operator to the most pertinent alerts.

1.1 Cyberspace as a Center of Gravity.

Carl Von Clausewitz [25], an early 19th century military strategic theorist, originated the concept of centers of gravity (COG), as a "hub of all power and movement upon which everything depends". The DoD's increased reliance upon cyberspace for accomplishing both support and operational tasks arguably positions cyberspace as its most critical COG. Cyberspace permeates air, land, sea and space operations. Cyberspace systems are force multipliers; by automating tedious administrative tasks and increasing the depth and breadth of day to day information sharing, military personnel are able to accomplish "more with less". Command and control (C2) systems provide synergistic effects, whether through advanced workflow systems for coordinating and disseminating orders, or by enhanced heads up display systems for tactical, operational and strategic decision makers. In a time of war, the GIG's massive bandwidth enables unprecedented speed for providing enhanced situational awareness capabilities such as unmanned aerial vehicle or precision guided munitions video feeds. Not only a critical enabler to joint warfare, the projection of cyberspace power for coordinated efforts provides an awesome advantage over less technologically advanced combatant forces. The joint warfighter's ability to get the right information to the right people at the right time in a trusted means is critical to deterring war and, when necessary, projecting power to protect national security interests.

1.2 Cyberspace Vulnerabilities.

The more complex and interconnected/interdependent systems become, the harder it is to secure them. Each new entry point, whether physical or virtual, introduces a set of potential vulnerabilities which must be hardened against malicious use. To make matters worse, it could take as little as just one vulnerability in the collective to provide the means by which an attacker can cause massive damage.

Even as the number and complexity of ubiquitous computing platforms are on the rise, software development has been increasingly outsourced. At the end of fiscal year (FY) 1994, the United States Air Force had 2,892 "blue-suit" (USAF military) computer programmers. By the end of FY 2008, the Air Force was down to 921 blue-suit programmers, one-third of its 1994 population [2]. Almost all of the hardware and software systems the Air Force depends upon to accomplish its mission are acquired as commercial-off-the-shelf (COTS) solutions. Desktop computers, network equipment and mobile computing devices, server software, operating systems and office automation suites are each purchased in this way. The Air Force lets countless contracts to build, install and maintain custom software solutions, in the form of stand-alone, client/server or internet applications in an effort to deliver long-term monetary savings to the government.

The problem with this trend with regard to systems security is that one does not know for sure that the software is built to do only what it is intended to do. How much does one know about another's development process? Do they follow secure software development best practices? Does one know if these hardware and software platforms are introducing (intentionally or unintentionally) vulnerabilities to be exploited? Industry experts [41] say that a person cannot trust the security of software they did not develop themselves. The trusted computing initiative attempts to address these problems, but it has been shown that the security of a computer system is undecidable at best since it is unknown what emerging vulnerabilities linger on the horizon. Put another way, threat protection is highly experiential – an computing system cannot protect itself against threats it has never encountered or anticipated.

1.3 Threats to Cyberspace.

Unfortunately, adversaries know of the DoD's reliance upon the cyberspace domain and its vulnerabilities, and is working hard to level the playing field and even to gain the advantage through information warfare tactics. Without engineering cyberspace to assure confidentiality, integrity, availability (CIA), the DoD loses a critical asymmetric advantage over a less advanced combatant force. Cyber threats have a goal to disrupt, degrade or deny the CIA of DoD computer networks. All-too-common attack methods include distributed denial of service (DDoS) attacks, deployment of viruses, spyware, rootkits or "botnet zombies", and interactive exploitation (hacking) of known software/hardware vulnerabilities such as causing buffer overflows to gain control of a computer system. Once a system is compromised, it can be used by the attacker as a jumping off point to another, deeper network attack, or to pilfer information for malicious use.

1.4 Threat Detection.

Intrusion detection systems (IDS) play a major role in cyberspace threat detection. An IDS is a software or hardware system which automates the process of monitoring events occurring in a computer system or network, analyzing them for signs of security problems [13]. The two main types of IDS are signature based IDS and anomaly detection IDS.

A signature-based IDS performs well for detecting known threats, but cannot properly identify a novel threat; it must have a signature in its database which exactly matches a given threat in order to detect it. System administrators must constantly update threat-detection signatures to keep up with the ever expanding threat pool. Additionally, detecting a network attack solely through signature-based systems has been shown infeasible [38]. As quickly as a signature is created for an exploit, a hacker can unleash a slightly different version, a variant, requiring a completely new signature for detection. This is why zero day exploits are widely successful; a signature has either not been created, or cannot be deployed in time to prevent all possible damage.

Given the DoD's heavy dependence upon cyberspace, the vulnerable nature of cyberspace, and the multitude of threats which aim to undermine its confidentiality, integrity and availability, continued research of this volatile domain is paramount.

1.5 Research Overview

1.5.1 Problem Domain. One major difficulty facing the cyber situational awareness research community relates to the massive dimensionality of the threat search space. Due to the expanse of the cyberspace domain and the limitations of processing power, the environment is only partially-observable. Stated otherwise, it is infeasible to permute all combinations of sensor data in real time, sensors need to be wisely chosen for the system to attend to. Finding an optimal feature subset for building a classifier has been shown to be an intractable problem [43], and many problems related to feature selection have been shown to be NP-hard [51]. Theoretically, given infinite time, sensors, storage and computing power, perfect situational awareness can be derived. Of course this is not the case; any situational awareness obtained will be constrained by the resources allocated and the time bounds acceptable to solving the problem. Situational awareness is concerned not only with its effectiveness, but also its efficiency.

1.5.2 Past Research Summary. Given that signature-based sensors are not feasible for detecting all threats, researchers must consider alternative solutions. Researchers have proposed threat detection methods, attacking the problem from various angles over the years; a representational few of these are discussed further in Chapter II. The bulk of the research efforts for threat detection thus far focuses on developing methods relying solely on network traffic [16] [46] [47] [11] [36] [38], solely on event logs [52] [69] [63], or solely on system calls [34] [55]. Surprisingly, no research efforts thus far has attempted to combine data from various system sensor categories, such as file I/O, network traffic and process meta data, in order to form a larger picture of what data is most relevant for the identification of threats. The fusion of multi-sensor data *is* studied at great length [15], but not toward the understanding of what data is most important to identifying specific cyberspace threats. Situational awareness research aims to bring together raw data to formulate higher levels of understanding. This requires transforming the data from its raw form into data which provides decision-quality information.

1.5.3 Problem Statement and Hypothesis. A problem which continues to plague intrusion detection systems is a high incident of false positive alerts, the alerting of malicious activity when it is not actually present. A system administrator must filter through these false alerts in order to find and act upon alerts which pertain to truly malicious activity. In order to improve the accuracy of an IDS and ease the burden on system administration personnel, a methodology for reducing the incidence of false positive alerts, while still accurately identifying malicious events, is needed.

This thesis presents a methodology to expand cyberspace data understanding for the purpose of improving threat detection accuracy. By identifying relevant features from an array of sensor data elements, this methodology identifies not only if a system is under attack or not, but also what stage of an attack is occurring.

The research hypothesis is that the discovery of correlated forensic evidence in data to a known attack event, the most relevant sets of features to attend to can be identified. Knowledge of which sets of features an IDS should attend to can greatly decrease the incidence of false positive alerts while still accurately, and economically, identifying malicious events.

1.5.4 Experiments. A small network laboratory is established consisting of a two machines: a target machine and an attack machine. The target machine is outfitted with a number of open source sensors to monitor network activity, system events and changes in process meta data. The attack machine is loaded with a number of malicious tools which can be used to scan the target machine for vulnerabilities and launch exploits to take advantage of these vulnerabilities.

Data is collected by activating the target machine's sensors and executing a scripted series of events. These events simulate normal, scanning and exploition activities as the target machine's sensors capture data to a repository.

A Java-based framework is developed to parse and consolidate thousands of sensor-generated files into a raw set of features to analyze, aggregated into two second time windows to allow for data alignment. The framework is built to be extendable, allowing for the incorporation of new sensors based upon new or changing data requirements.

A Microsoft SQL Database is used to aggregate events into two-second timeframes, and to analyze both continuous and discrete features. Numeric features are compared statistically, to include differences between min, max, mean, variance, skewness and kurtosis. A comparison of the statistical differences between normal, scanning and exploitation activity is performed. Numeric features are only selected if their measures are significantly outside those of normal activity. A set selection method is used to analyze discrete features. Features which are in either the scanning or exploit collection, but not in the normal collection (or occur in much higher aggregated incidence than in the normal collection) were correlated against experiment events at the time. Selected features were used to label data as normal, scanning or attack.

A MATLAB neural network is trained and then used to classify, in an unbiased manner, records of the selected features. The performance and classification accuracy of the neural network is analyzed. Labeled data was manually cross-referenced against the scripted events to determine false positive and false negative alerts the feature set generated. Results were compared to alerts generated by a leading open source IDS.

1.5.5 Assumptions and Limitations.

The following are assumptions and known limitations of this research:

Controlled Environment For research and analysis purposes, and due to the potential damaging consequences of performing network attacks on a "live" network, experiments are carried out in an isolated computer lab environment. Experiments use pre-scripted simulations of a small subset of normal, scanning and exploit activity, and do not represent the totality of activities which can be executed on (or against) a host machine. Collections are additionally focused on monitoring one system, not a series of systems or their network. Due to these facts, resulting feature selection and classification data is considered biased toward the operating system monitored during experimentation, selected attack tools and sensors, and the events in the scripts. However, since this thesis applies a methodology which can be extended, it serves as a proof of concept versus a deliverable IDS platform. Additional collections across many systems including varied events will likely produce different results.

Sensor Impact To System The selected methodology involves the collection of a live system's internal forensic data, which involves execution of programs and persisting data to the target system. While care is used to select lightweight sensors over more burdensome ones, sensor activity impact collections in two ways. First, the system being monitored is running additional programs to sense the host environment, which utilizes system resources such as the machine's processor(s), memory, and hard drive. Second, collected data reflects information pertaining to sensor activity. These facts should be taken into account during later stages of the experiment.

Partial Observable Environment Given the size of the search space and the limitations of processing power and the sensors selected, the host environment is only partially observable. Stated otherwise, it is infeasible to observe all possible portions of system's state in real time. Given this fact, care is taken to choose sensors which both provide a unique view of the search space in order to provide as much forensic evidence as possible. Optimality conditions for sensor selection include consistency, completeness, speed, and resource overhead. While no formal method is used to select sensors for observing the target environment, these conditions were taken into account.

1.5.6 Results and Conclusions. Results show that the methodology presented in this thesis improved classification of host-based events by 91% fewer false positive/false when compared to a leading open source intrusion detection system for a set of controlled attack experiments. The improved accuracy not only aids in the identification of malicious events, but also reduces administrative burden for network defenders who would have to manually filter through false alerts to find truly relevant events. The application of the CRISP-DM process is an effective means by which to discover hidden data relationships within forensic evidence.

II. Literature Review

To support this body of work, an understanding of related work in the area of the threat detection problem, as it pertains to cyberspace situational awareness, must be developed. Pertinent areas of research within the scope include, but are not limited to: intrusion detection systems, cyberspace situational awareness systems, knowledge discovery and artificial intelligence.

As a nation and a profession of arms, the United States and USAF rely heavily upon the cyberspace domain. However, time and time again, one learns all too painfully of the vulnerabilities and varied threats to cyberspace. Hundreds of solutions have been applied to the threat detection problem; development of data understanding is one method in a consistently growing area of intrusion detection research.

2.1 The Cyberspace Domain

The Cyberspace domain is characterized by "the use of electronics and the electromagnetic spectrum to store, modify, and exchange data via networked systems and associated physical infrastructures" [24]." As mentioned in Joint Publication (JP) 6-0 [7], *Doctrine for Command, Control, Communications, and Computer (C4) Systems Support to Joint Operations*, the Global Information Grid (GIG) is "the globally interconnected, end-to-end set of information capabilities, associated processes and personnel for acquiring, processing, storing, transporting, controlling, and presenting information on demand to joint forces and support personnel." The GIG includes "all owned and leased communications and computing systems and services, software (including applications), data, security services, and other associated services necessary to achieve information superiority." Cyberspace is utilized across air, land, sea and space domains. In other words, the GIG is the infrastructure upon which all digital operations the Air Force and its partners rely upon to carry out the mission, from administrative tasks to full-scale operations.

2.1.1 Cyberspace Priorities. There are four strategic priorities listed by the Chairman of the Joint Chiefs of Staff related to Cyberspace [24]:

- Gain and maintain initiative to operate within adversary decision cycles.
- Integrate cyberspace capabilities across the range of military operations.
- Build capacity for cyberspace operations.
- Manage risk for operations in cyberspace.

While this work pertains to each of the Chairman's four priorities, it is most relevant to the latter two. First, the *capacity for cyberspace operations* relies heavily upon gaining and maintaining information superiority. The methodology demonstrated in this work can be applied toward enhancing information superiority, specifically with regard to host-based intrusion detection. Second, the expansion of cyberspace data understanding through this and similar works aid in the detection of threats, risky software or computing practices, which can be applied to *managing risk for operations in cyberspace.*

2.1.2 Cyber Center of Gravity (COG). As mentioned in Chapter 1, Carl Von Clausewitz [25] originated the concept of centers of gravity concept as a "hub of all power and movement upon which everything depends." JP 1-02 [5], *Doctrine for Joint Planning Operations*, further defines a center of gravity as a "source of power that provides moral or physical strength, freedom of action, or will to act." JP 5-0 [6], *Joint Operation Planning*, states that the most important task confronting planners in this process is "being able to identify friendly and adversary strategic centers of gravity; that is, the sources of strength, power, and resistance. This task is critical because "faulty analysis of friendly or adversary centers of gravity can have very serious consequences; specifically, the inability to accomplish the military objectives at an acceptable cost and the unconscionable expenditure of lives, time, and materiel in efforts that do not produce decisive strategic or operational results.

Military theorists have identified four major powers, or centers of gravity, which a nation state can use to influence another nation state:

- Military power: Capability to apply military arms to project violence.
- Political power: Diplomatic capacity and public support.
- Industrial: Ability to manufacture and sell goods
- Economic: Monetary reserves (and currency valuation within global market).

JP 6-0 [7] states that command and control (C2) consists of two elements: people, and (collectively) C2 facilities, systems, communications, procedures. JP 6-0 also states that "C2 is essentially about information: getting it, judging its value, processing it into useful form, acting on it, and sharing it with others." Of the two basic uses for information, the first is to "help create situational awareness (SA) as the basis for a decision." The second is to direct and coordinate actions in the execution of the decision.

Certainly, the Cyberspace Domain provides a source of strength, power and resistance for the DoD. It is a critical COG, as cyberspace permeates throughout the DoD mission. It may also be the DoD's most vulnerable COG: like the space domain, cyberspace has no definitive borders. However, unlike space, cyberspace is accessible at a very low cost, and can be exploited from any one or a group of millions of interconnected phones, radios and computers. Unfortunately, our enemies know this as well, and are actively pursuing ways to undermine this most critical center of gravity. The DoD's most important asset, without regard to its people, is its information. The DoD operates networks with varying levels of security classification.

2.1.3 Vulnerabilities.

Computers and networks expose a number of vulnerabilities which can be exploited by malicious parties. Among these are physical, system/network, personnel, and software vulnerabilities.

Physical Vulnerabilities. Lack of physical protection of computing assets and information is the most basic vulnerability. When discussion about physical vulnerabilities arises, one typically thinks of physical access to business office, network operations control area and server room security. However, with today's pervasive and ubiquitous computing capabilities, physical vulnerabilities are now extended into personal residences, automobiles and even on a person. A laptop or mobile computing device left unsecured from theft can present a golden opportunity to the would-be intruder. The Washington Post [56] reported on a disturbing trend in Chicago indicates up to 160,000 mobile devices (cell phones, blackberries, etc) are left behind in city taxi cabs, but only 50–60% of those ever get reunited with their owner. They further cited a related survey done by Symantec which indicated that up to 37 percent of those devices potentially contain confidential business data; in Chicago alone, this would result in 26,640 potential breaches in security. Examples of breakdowns in physical security potentially affecting military personnel include the now infamous 2006 Veteran's Administration "stolen" laptop which contained privacy act records of 26.5 million past and present U.S. service members, and the 2009 discovery [26] of an MP3 player purchased for $9 from a pawn shop which contained personnel data and mission briefs from active theaters of operations. An unsecured asset which falls into the wrong hands can yield disastrous consequences, from identity theft to operational compromise which can result in the loss of human life.

Personnel Vulnerabilities However, in the hands of an untrained, negligent, or malicious user, these protections can easily be incorrectly configured, disregarded or potentially subverted. If users improperly install, configure or maintain a system, they expose their systems to vulnerabilities that could have been otherwise mitigated. Some examples of these negligent acts include not changing the default password for administrator accounts, not disabling/deleting guest accounts, not setting web browser security settings appropriately, not keeping system patched against newly discovered vulnerabilities, implementing weak password policies, and utilizing no/weak encryption. Examples of malicious user activity are discussed in Section 2.1.4.

Computer System/Network Vulnerabilities. A computer typically has an operating system such as Microsoft Windows or Ubuntu Linux, office automation software like Microsoft Office or OpenOffice suites, application software such as Mozilla Firefox or Adobe Illustrator, antivirus and/or personal firewall software, entertainment and/or other software. Collectively, thousands of files compete for resources and communicate internally with each other and across networks. Particularly with operating systems and applications which communicate across a network, many programs have user-configurable protection mechanisms intended to provide users the capability to minimize their exposure to vulnerabilities. Likewise, networking hardware such as routers, switches and firewalls each have software which set rules for systems, ports and protocols allowed to communicate across them.

Software Vulnerabilities. Computer systems communicate with each other across shared infrastructure using standard protocols. Because of the open standards communication systems use, it becomes imperative to ensure only safe (and authenticated as appropriate) communication takes place, and programs should properly handle any exceptions that arise from malformed inputs. Unless a computer system's programs are developed with security in mind from the beginning, they likely contain exploitable vulnerabilities. Malicious user inputs to such programs can subvert the security of the system, allowing an attacker to disrupt, deny or degrade the confidentiality, integrity or availability of the victim's system. Howard, et al. [41] describe in great detail the nineteen "deadly sins" of software security, as summarized in Table 2.1.

Table 2.1: 19 Deadly Sins of Software Security [41].

19 Deadly Sins of Software Security	
Vulnerability	**Coding Sin**
Buffer Overruns	Failure to validate array boundaries before read/write

Continued on Next Page…

Table 2.1 – Continued

Vulnerability	**Coding Sin**
Format String Problems	Failure to validate user input for malicious format string sequences before returning data
Integer Overflows	Failure to check for integer over/underflow conditions when performing mathematical operations or casting data types
SQL Injection	Failure to validate user input for malicious SQL statements
Command Injection	Failure to validate user input for malicious command statements
Failing to Handle Errors	Failure to ensure errors are appropriately handled
Cross-Site Scripting	Failure to validate user input for malicious HTML
Failing to Protect Network Traffic	Failure to use secure protocols for communication
Use of Magic URLs and Hidden Form Fields	Failure to avoid using magic URLs and hidden form fields to transmit sensitive information
Improper Use of SSL and TLS	Failure to validate certificates, chain of trust and check certificate revocation
Use of Weak Password-Based Systems	Failure to strongly encrypt passwords for transmission and/or storage
Failing to Store and Protect Data Securely	Using weak access control mechanisms or hard-coding secret data
Information Leakage	Failure to protect sensitive information from inadvertent disclosure
Improper File Access	Failure to verify user access to files or protect against directory traversal attacks

Continued on Next Page...

Table 2.1 – Continued

Vulnerability	**Coding Sin**
Trusting Network Name Resolution	Reliance upon Domain Name Service or similar service, which may be compromised or spoofed, for establishing connections
Race Conditions	Failure to avoid race conditions between threads or processes
Unauthenticated Key Exchange	Failure to validate sender identity during key exchange
Cryptographically Weak Random Numbers	Using pseudo-random number generation for cryptography
Poor Usability	Designing a non-intuitive or overly complex user interface, especially for setting security features

2.1.3.1 Hacking Tools. There are thousands of tools hackers can use to discover systems and their vulnerabilities, crack passwords and exploit system security deficiencies. Over the years, system administrators developed tools for centralized or remote troubleshooting and administration of network systems and user accounts. Unfortunately, in the hands of a hacker, these tools provide invaluable insights which can be used for malicious ends. A few of the prominent tools as listed by Insecure.org [8] are highlighted in Table 2.2.

2.1.4 Threats. Lt. Gen. William Shelton, the Air Force's Chief Information Officer, recently briefed the House Armed Services Committee panel, "Threats in cyberspace move at the speed of light, and we are literally under attack every day as networks are constantly probed and adversaries seek to exploit vulnerabilities." Threats come in many forms, some from direct interaction by hackers, others via programs or tools developed by hackers. Some of the main threats to the confidentiality,

Table 2.2: Common tools used by hackers to discover host systems, enumerate and exploit their vulnerabilities.

Google	The ubiquitous search engine of choice for hackers can be used to obtain corporate knowledge, akin to dumpster diving, to learn things such as employee names and unsecured sensitive information about communications systems and programs
Nmap	Program which performs host and port scanning to discover machines and services running on a network
Dsniff	Suite of network auditing and penetration testing programs
Nessus	Vulnerability assessment tool for Unix and Windows
Wireshark	Network protocol analyzer for Unix and Windows
Metasploit	Framework for developing, testing and deploying exploit code
Tcpdump / Windump	Network packet sniffer for Unix / Windows
Cain and Abel	Password cracking tool for Windows
Ping	Tool which tests ability to reach a specific machine by IP Address
Netstat	Displays protocol statistics and current network connections
Tracert	Displays network route between querying machine and target machine by IP Address
SSH	Secure terminal emulator for remotely interacting with command line
SysInternals	A suite of Microsoft Windows tools (ProcessExplorer, PsTools, Autoruns, RootkitRevealer, TCPView among others) for monitoring activity on a system.

integrity and availability of DoD networks include malware, denial of service attacks and insider threats. At the root of each of these threats is the hacker.

2.1.4.1 Hackers. An understanding of the hacker mindset, to include motivations and how hackers view systems, is important to establishing effective detection techniques. Gregg, et al. [37] discusses various types of hackers, thier motivations and varying levels of expertise. There are three main categories of hackers: whitehat, blackhat and grayhat. Whitehat hackers perform "ethical hacking": they employ hacking techniques to uncover security vulnerabilities, and use this information to better secure their networks. However, blackhat hackers have the intent of exploiting security vulnerabilities for malicious ends (disrupt, deny or degrade service, unauthorized data access or exfiltration, financial theft, etc). Grayhat hackers do a little of both, have ambiguous loyalties and should not be trusted for whitehat duties.

Blackhat hackers tend to be either means-oriented or goal-oriented.

Means-Oriented Hackers. Means-oriented blackhats are hackers which focus on breaking a particular type or facet of a computer system, and include phreakers, whackers, software crackers/hackers and system crackers/hackers. Phreakers are the original hackers, who took advantage of weaknesses in telecommunications systems with a simple goal: making free phone calls. Whackers are classified as blackhat hackers who typically focus their skills on attacking wireless networks. Software crackers/hackers typically use reverse engineering to bypass software licensing, but may also use their skills to subvert other software security mechanisms (such as encryption) for further exploitation. System cracker/hackers are highly skilled at attacking vulnerabilities in operating systems; they build worms, viruses and trojans with the potential for world-wide impact.

Goal-Oriented Hackers. Goal-oriented blackhats focus more on the ends: they desire computer system compromise in order to accomplish some objective. Goal-oriented blackhats include script kiddies, disgruntled employees, hacktivists and cyber terrorists/cyber-criminals. Script kiddies are those blackhat hackers who run pre-built hacking programs (by hackers with better skills) against a target system; they tend to have no or minimal computer programming experience or skills, and most likely do not understand how the tools they use work. This does not diminish the threat they pose, however. Script kiddies have been known to, among other things, steal (and use) credit card and banking account information and access classified government computer systems. A disgruntled employee presents a particularly dangerous threat, as they have not only already been granted access to systems for the performance of their duties, but an increased potential for employing social engineering techniques as a "trusted agent" of the organization to gain information to further exploit systems. Hacktivists are blackhat hackers who have a political motivation to attack a system. Hacktivists generally deface websites to promote their political ideology. Cyber terrorists/cyber-criminals are those blackhat hackers who are paid to attack or exploit assets belonging to governments, corporations or individuals [37]. Their goals are to generally to exfiltrate sensitive information (e.g. governmental/corporate espionage) data or to disrupt or deny access to resources.

2.2 Attack Process

A hacker cannot exploit a computer system he/she knows nothing about. In fact, the process to exploit a system is typically multi-phased, and can occur very quickly (and noisily from a sensor perspective) or span days or weeks (and stealthy from a sensor perspective, "low and slow" to avoid automated detection). In order to exploit systems on a network, a hacker must first learn what systems are reachable, what services are hosted by the reachable systems, and discover potential vulnerabilities through the identification of installed software versions and patch levels. Some of these steps may be bypassed due to data gleaned from dumpster diving, web surfing or social engineering tactics. Depending upon the goals of an attack, and the knowledge the attacker has on hand, an attacker will carry out one or more of the following stages of attack processes. McClure, et al. formally categorize attack stages into

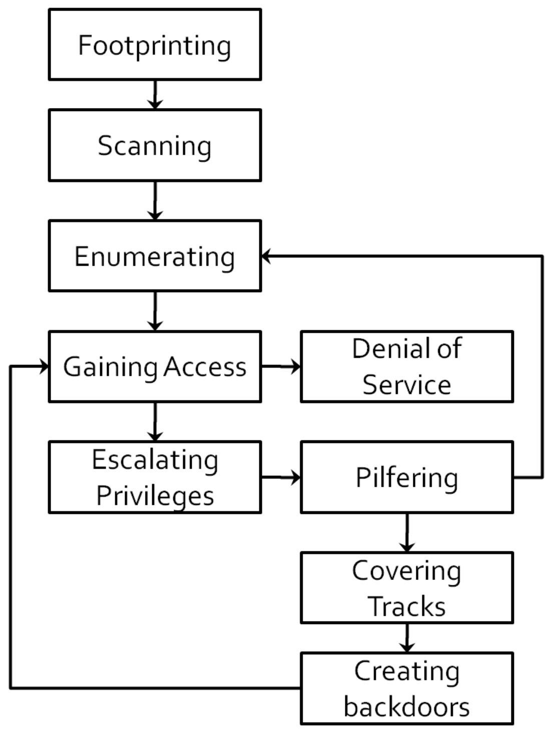


Figure 2.1: McClure, et al's. "Anatomy of a Hack" (adopted from [53]).

footprinting, scanning, enumeration, gaining access, escalating privilege, pilfering, covering tracks, creating back doors, and denial of service activities; the process flow is depicted in Figure 2.1 [53]. Gregg [37] offers an almost identical process, but uses the term reconnaissance in place of footprinting, and does not address pilfering and denial of service as formal stages of the attack process.

2.2.1 Footprinting. The first stage of an attack process, footprinting, is to farm through publicly available information to build a profile of the organization's security posture. This farming is typically aimed at learning ways to intrude the organization's internet, intranet, extranet and remote access environments [37]. By using a variety of tools and data sources, a hacker can learn domain names, network blocks and subnets, DSN hostnames, system architecture, protocols used, and other clues that provide starting points within the environment to later map out the network [53]. To footprint a network, a hacker may start by querying information listed by domain registrars and DNS servers (e.g. via whois or nslookup), or may attempt to gain insights by web searching for postings linked to the organization (e.g. organization's public website, third party blogs, job or alternative sites, or via Google hacking [37]. Other tactics a hacker may use to discover information about an organization's environment include dumpster diving, web surfing or social engineering tactics such as phishing [53].

2.2.2 Scanning. The second stage of an attack process is scanning. During the scanning stage, a hacker scans a network to determine what systems are reachable. The attacker implements various ping utilities such as fping or hping; port scanners such as FoundStone's SuperScan or ScanLine programs, or operating system detection utilities such as amap or sinFP. Nmap, and its graphical version Zenmap, provides all of these scanning capabilities in one package. NetCat is a unix-based tool which provides these capabilities and more, but the Windows UDP scanner is unreliable [53]. Scanners work by sending basic ping requests in search of ping replies (for example, sending a ICMP ECHO request to elicit an ICMP ECHO REPLY). A reply indicates discovery of a host. Also, firewalls may block some of these types of requests as they are detected.

Once a hacker identifies hosts which are alive, they focus scanning efforts to look for open TCP and/or UDP ports on the systems. To do this, a hacker sends network packets to ports on the host machine in order to elicit the appropriate response. There are eight basic TCP scan types: TCP connect scan, TCP SYN scan, FIN scan, TCP Xmas Tree scan, TCP Null scan, TCP ACK scan, TCP Windows scan and TCP RPC scans. TCP connect scans are easily detectable, so hackers developed these other "connectionless" scans. Some of the scans work in a standard way based on their RFC, but others, such as the TCP FIN scan, TCP Windows scan and TCP RPC scan respond in an operating system-specific way or do not work at all on some operating systems. A UDP scan to an open port will result in no response and to a closed port will result in an ICMP port unreachable message. [53]

Hackers can determine what operating system is installed on a machine. Some of this “operating system fingerprinting” can be attained by observing which ports were open on the machine; a Windows host typically listens on ports 135 and 139. Fingerprinting is enhanced through sending a series of probes to open ports and monitoring the feedback received from the ports. For example, an attacker may send a FIN probe (a packet to an open TCP port with the FIN bit set). If the machine is Windows NT, Windows 200X or Windows Vista, it will respond with a FIN/ACK packet, which is not the standard response per RFC 793. Other probes look for the use of “don’t fragment bit”, the TCP initial window size, the length of ICMP quoted messages. Nuances in how an operating system handles TCP packets help an attacker to fingerprint the operating system. This information is invaluable to narrowing the scope of which vulnerabilities to search for.

2.2.3 Enumeration. Enumeration is the third stage of an attack, and is characterized as targeted queries to a host’s ports in order to learn more about what service is running on it. One way to enumerate services is through banner grabbing through utilities such as telnet or netcat. For example, by attempting to open and use a telnet connection to an open port discovered during the scanning step, a hacker can monitor feedback from the target system to glean information about the service running on the port (such as protocol and version). Nessus, a vulnerability scanner, goes a step further. Nessus scans a system’s ports to determine what services are running ion but also tell what specific vulnerabilities a system is susceptable to is the Nessus program. Nessus is a vulnerability scanner, which roots out , and can even Once a hacker finds a reachable system, the next goal is to determine which ports, protocols and services the system supports. This helps the attacker determine potential vulnerabilities. [53] Other enumeration techniques include querying ports with FTP, querying DNS information about a using nslookup (from a Windows machine), or using native Unix utility programs such as finger (against Unix-based targets) or ls -d ¡domain name¿ to query DSN zone transfer information about a network. Enumeration techniques exist for all sorts of specialized services such as NetBIOS, HyperText Transfer Protocol (HTTP), RPC, Lightweight Directory Access Protocol (LDAP), Structured Query Language (SQL); McClure lists enumeration for these services and more in [53].

2.2.4 Gaining access. The next stage of an attack is gaining access. Once a malicious party discovers a vlunerability on your system, there are many tools available exploit the weakness in order to gain access. Once such tool is MetaSploit [53]. A hacker need only browse through the MetaSploit vulnerabilities database through the user interface, select a payload, set an option or two (e.g. target machine and stealth settings), and the payload is launched as requested. Some payloads perform a specific action such as adding an admin account, others provide a reverse shell connection to allow the hacker to log on to the target machine with System privileges. At that point, the target machine is pretty much at the whims of the remote user, and the system should be considered completely compromised if this type of access is detected.

2.2.5 Escalation of privilege. If gaining initial access is not enough to accommodate the hackers goals, the hacker typically attempts to gain additional privilege by accessing password databases using tools like ophcrack, Cain or other password cracking tool, or by searching through files looking for passwords which provide additional access [53].

2.2.6 Pilfering. Depending on the hacker's goal, they may want to use their initial breach as a stepping off point to deeper penetration. Pilfering involves searching the network for additional passwords, trust relationships with neighboring networks, or system vulnerabilities that may not have been visible or exposed to the attacker prior to the breach. Tools such as Cain & Abel and simple file searching can yield just the type of insider information the intruder needs to gain further access [53].

2.2.7 Covering Tracks.

To prevent prosecution or to enable repeat visits without arising suspicion from network administrators, a hacker may desire to cover their tracks by attempting to erase the forensic evidence of their penetration. Covering tracks may involve clearing logs, altering logs, or hiding tools through file streams or rootkits [53].

2.2.8 Creating back doors.

An attacker may leave behind back door programs which eases access for a repeat visit or monitors a target (e.g. via keylogging). Simple measures such as scheduling a job to run after hours which opens a port on the computer, or replacing an application such as calc.exe with a trojan version [53].

2.2.9 Denial of Service.

If the goal of the attacker is to disrupt, degrade or deny service, they may choose to launch a denial of service attack. Such an attack may flood buffers, send malformed communication to cause a service to fail, overwhelm system resources via distributed denial of service attacks. According to Mcclure [53], "DDoS attacks are the most significant operational threat that many online organizations face today."

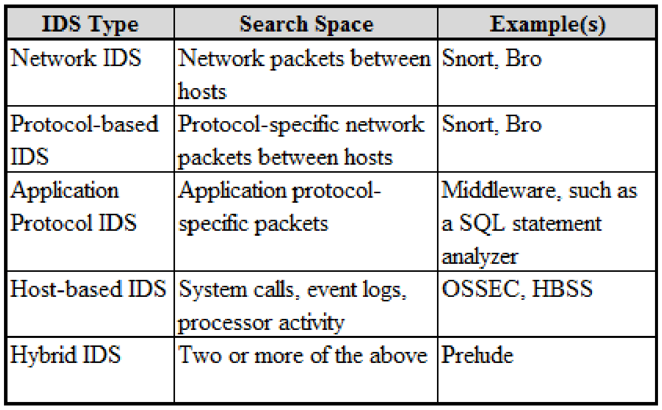
2.3 Sensors

In order to perceive elements within a target domain, sensors must be employed within the domain's environment which are capable of detecting and signalling the occurrence of events. In the air domain, for example, the FAA employs weather and flight radars, radios and a whole host of digital avionics sensors which are constantly gathering and reporting data to decision makers: pilots, flight towers and ground crews, in order to keep the skies safe. Likewise, sensors are employed in various ways throughout the cyberspace domain.

There are essentially three categories of sensors needed to perceive event features on a host in the cyber domain: files, input/output (I/O) streams, and processes/memory. Within operating systems, sensors take the form of software programs which monitor resources such as memory, the central processing unit (*CPU*), and network interface cards (*NIC*), reporting events such as page faults, network request timeouts, and input/output errors. Sensors are also used to monitor the security of the system, reporting events such as privilege escalation, successful or failed login attempts, and noting when services are started/stopped. Finally, applications often "sense" and log key application-specific events, such as when they are successfully started or stopped, or when they complete or fail a noteworthy task, such as a "SQL Server is now ready for client connections" informational message.

2.3.1 File Sensors. First, *file sensors* are needed. File sensors should provide the ability to inspect a file's contents and/or its properties, such as file size, its checksum or hash, the date/time last updated or accessed, file size, access control list, whether it is open or not, how often it is opened, what process/user typically opens or updates it, fragmentation level, password strength as applicable, etc. Files range from executable and data files such as notepad.exe or update.txt to operating system and special files, such as event log files, registry files, or security account manager files. Individual files may require highly specialized sensors or word parsers to extract meaning. Examples of this sensor category on a typical Microsoft Windows platform includes filemon, regmon, event viewer, or custom-built applications which enumerate Active Directory Service Interface objects, query databases, or execute various command line entries for directory listings and the like.

2.3.2 I/O Stream Sensors. Second, input/output (I/O) stream sensors are needed. Typically, one thinks immediately of a network interface card (NIC) sensor, such as Wireshark, for monitoring network traffic being transmitted to or from a system or subnet. But there are numerous other I/O devices on a host, such as USB devices, keyboards, mice, graphic cards, etc. These sensors should provide the ability to inspect the I/O traffic contents and/or its properties, such as packet size, protocols used, source and destination for messages, type of message, size of message, or collective properties such as number of messages/second, volume of traffic, etc.



IDS Type	Search Space	Example(s)
Network IDS	Network packets between hosts	Snort, Bro
Protocol-based IDS	Protocol-specific network packets between hosts	Snort, Bro
Application Protocol IDS	Application protocol-specific packets	Middleware, such as a SQL statement analyzer
Host-based IDS	System calls, event logs, processor activity	OSSEC, HBSS
Hybrid IDS	Two or more of the above	Prelude

Figure 2.2: Types of intrusion detection systems and their related search spaces. [17]

2.3.3 Process and/or Memory Sensors. The third category of host-based sensors needed is *process and/or memory sensors*. These sensors should be able to enumerate the system's memory contents, and/or its properties, such as memory addresses, memory allocation, associated process(es), page fault rates, process owner, number of threads, processor time, etc. Examples of this sensor category include process monitor, task manager, and performance monitor.

2.4 Intrusion Detection Systems

In 1987, according to Denning, et al., [30] an intrusion detection system (*IDS*) monitors network communication to determine whether activity is unusual enough to suspect an intrusion. However, as Black highlighted [17] twenty years later, and as covered in Section 2.4.1, there are now several types of IDS. An IDS can be deployed as a hardware and/or software solution and either use signatures or heuristics for detecting malicious threats, as discussed in Section 2.4.2.

2.4.1 Types of IDS. There are several types of IDS [17]: Network IDS, Protocol-based IDS, Application protocol-based IDS, Host-based IDS and Hybrid IDS; each provides a layer of protection, as summarized in Figure 2.2. A brief description of each is provided.

A network IDS (NIDS) analyzes network packets as they are transmitted between systems [28]. Typically, NIDS are positioned at network boundaries, monitoring traffic for malicious activity as it enters or leaves a network. When anomalies are detected, a NIDS can take various actions, such as simply logging the event, alerting administrators or coordinating with a firewall to deny the anomalous traffic. If properly configured, a NIDS is capable of detecting threats originating from outside or from within the network such as (but not limited to) DNS spoofing, TCP hijacking, port scanning, distributed denial of service and data exfiltration [28].

NIDS has a number of shortfalls when it comes to detecting intruder threats. First, if an attacker obfuscates or encrypts the data they are transmitting, the NIDS is reduced to analyzing packet headers only; the NIDS' ability to detect malicious activity within payload data is severely hampered at best [22]. Second, when it comes to insider threats, the NIDS would serve as a last line of defense for traffic leaving the network (and may not be configured to detect host-to-host activity within the network). However, before network traffic is generated by an insider threat, there could be readily available tell-tale signs within the affected computer(s) that anomalous activity has occurred. Since a NIDS only monitors network traffic [28], it has no access to the internal state of the surveilled machine (e.g. memory, process, file or registry activity and system, security or application logs) of individual machines on the network. A more introspective approach is needed for early detection of insider threats within the malicious activity's sequence of events.

A leading open source NIDS, Snort [64], is available for download from snort.org. Snort was first released in 1998, and due to its open source nature, has had hundreds to thousands of contributors refining the product and its rules ever since. With twelve years worth of research and development, it is considered the de facto standard in intrusion detection and prevention [64].

A Protocol-based IDS (PIDS) and application protocol-based IDS (APIDS) are each similar to NIDS: they both monitor streams of inbound and/or outbound data.

A PIDS analyzes network packets traveling between systems via a specific port or protocol. [28] For example, a web server system administrator may wish to employ a PIDS to analyze traffic coming across port 80 (HTTP) or port 443 (HTTPS), looking for malformed or malicious web-based queries such as mentioned in [41]: attempts to cause buffer overflows via lengthy URL strings, attempts to inject execute arbitrary code using magic URLs, or attempts to exfiltrate sensitive files from the web server through directory traversal. NIDS tools such as Snort and Bro [44], perform protocol analysis. An APIDS analyzes the communication via application-specific protocols. For example, an APIDS could be loaded as middleware between a web server and a back-end database server, analyzing SQL requests and responses for SQL injects and cross-site scripting attacks [17]. Both PIDS and APIDS are limited scope intrusion detection systems. They can protect portions of a system, but will not detect attacks that happen outside their observable spaces.

A host-based IDS (HIDS) analyzes the *activities* of internal mechanisms [17] of a single machine, e.g. its processes, memory, files and event logs to determine the presence of anomalous activity (as opposed to a traditional antivirus program, which looks for malicious signatures in files, email and memory [1] for malware). Specifically, a HIDS monitors interactions with the host operating system, looking for such things as abnormal processes or user activities which indicate the presence of malware, unauthorized privilege escalation or other threats. To detect these types of threats, raw forensic data observations such as system call sequences, file system modifications, and user logon traces are preferred over higher-level data like event logs whenever possible. The lower the level of data being analyzed, the deeper the penetration has to be to avoid detection .

Defense Information Systems Agency's (DISA) Enterprise-wide Information Assurance and computer Network Defense Solutions Steering Group (ESSG) procured a Host-based intrusion prevention system / host based firewall solution, which is dubbed Host-Based Security System (HBSS). A commercial-off-the-shelf (COTS)-based application written by McAfee, HBSS is to be installed on each workstation and server in the DoD.

A hybrid IDS combines one or more of these IDS approaches. [17] While a standalone HIDS has no access to another computer's internals, some HIDS are configurable for distributed defense, using information sharing to strengthen the collective defensive posture. A NIDS teamed up with multiple host-based IDS is considered a hybrid IDS, such as the open source project named Prelude [71].

2.4.2 Signature-based IDS vs Anomaly Detection. There are two main detection methods an IDS employs: signature detection or anomaly detection. Each method has its pros and cons, which is discussed.

With signature-based IDS, observed data is compared against tersely defined rules modeled after known malicious activity. An IDS which employs this technique can only be effective at detecting threats which it has been programmed to detect. Examples of signature-based IDS methods include state transition modeling [42], and expert systems which employ string/rule pattern matching [11].

There are a couple of main shortcomings of employing a signature-based IDS. First, a signature-based IDS is unable to detect novel threats (for which it has not been programmed), or even variants of known threats [46]. As quickly as a threat signature is created and deployed, tens or hundreds of threat variants could be created or unleashed. A signature-based IDS must be constantly updated with the latest signatures to keep up with the most current threats.

The second shortcoming has to do with the feasibility of searching for known vulnerabilities in near-real time. The number of known viruses discovered over the last 20 years, as depicted in Figure 2.3, indicates an exponential growth trend. As of May 25, 2007, this figure was just over 300,000 [67]. In April 2008, a Computer World interview with industry experts [27] unveiled that number would surpass 1,000,000 viruses in 2009. While more recent figures are not yet available about how many viruses exist, this latter projection supports the 20-year trend. The implication this

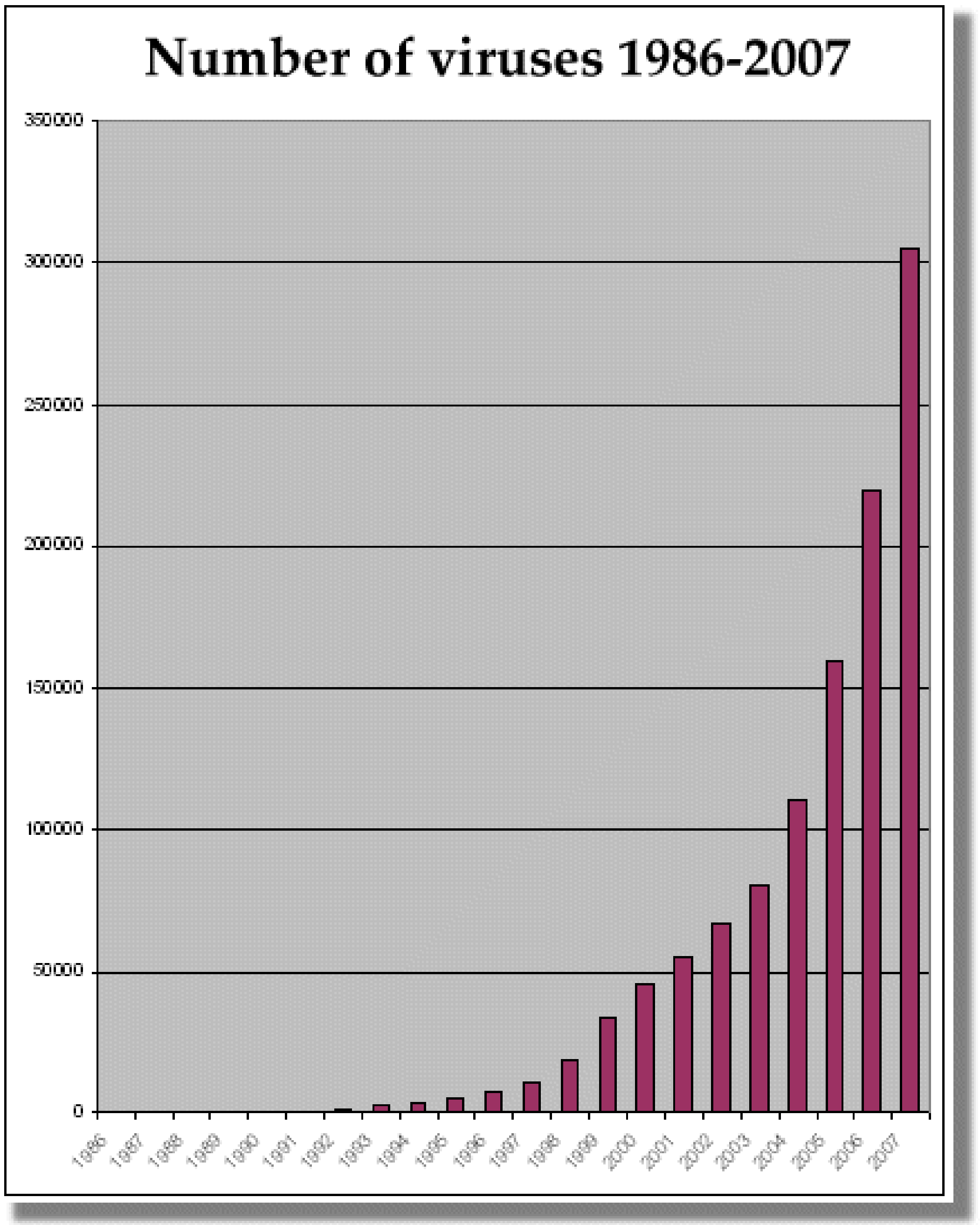


Figure 2.3: From 1986-2007, the number of known viruses indicates an exponential growth trend [67].

has to a signature-based solution is that scanning against a signature database will continue to take longer and longer, to the point of infeasibility in near-real time.

Anomaly detection involves using machine learning to train statistical models, or classifiers, to recognize malicious or anomalous patterns, or classes, of activity. A model is composed of a set of distinguishing features within the data which forms a pattern to be recognized. The recognition of a pattern allows the classifier to properly categorize data as one class of data or another. Applied to the IDS problem, one needs to distinguish (at minimum) between normal and a malicious classes. A classifier can be taught to recognize more specific classes of data as well, e.g. the attack vector, or the type of normal user activity. Subclasses of this data can be defined with more specific statistical models. This approach can be effective for detecting novel threats depending on the strength of the classifier's underlying model [48].

Anomaly detection algorithms are trained via one of three types of machine learning: supervised, unsupervised and reinforcement learning. With supervised learning, an expert provides labeled data to train the classifier directly. With unsupervised learning, the system learns on its own what is considered normal activity, and flags everything else as anomalous [11]. Unsupervised learning is considered more appropriate than supervised learning for traffic classification because it does not rely on pre-defined classes [16]. Reenforcement learning falls in between supervised and unsupervised learning. With reinforcement learning, the classifier starts out trying to figure out on its own what is considered normal activity, but is rewarded or penalized by an expert for how effective the classifier was. This feedback is fed into the classifier to enhance refine its model.

Like its signature-based counterpart, anomaly detection is not without faults. Since statistical models are heuristic measures, and not perfect indicators, they result in errors in the form of false positives and false negatives [17]. False positives classify normal activity as malicious activity, which would then typically trigger a manual review process by an system security expert. False negatives classify malicious activity as normal behavior, which would allow an intruder's activity to go undetected. The whole anomaly detection game is to train a model which minimizes both types of errors. There are three additional factors which cause problems for anomaly detection.

First, anomaly detection models are highly experiential. Since an exhaustive training set is infeasible due to dimensionality, there will be malicious activity the classifier has never been exposed to. This results in undecidable events which can be misclassified by the underlying statistical model. Additionally, a model can become biased if it is "overfit" to noisy data [49]. A model which overfits to a particular class of data accepts less variance when trying to recognize said class in future data. Stated another way, data of a particular class which does not closely match the trained model is more likely to be misclassified as another class of data.

The second challenge with anomaly detection IDS is that malicious activity can sometimes (intentionally or otherwise) closely resemble normal activity, close enough that it does not appear anomalous to the classifier. Such is the case with "low and slow" attacks, which fly below the radar of an IDS, against a computing target. Some of these activities can be better distinguished by modifying the statistical model; for example one can change which features comprise the model or add weighting to features within the model to increase its overall effectiveness.

Third, anomaly detection classifiers may require maintenance similar to its signature-based counterpart, in the form of periodic (or possibly continuous) re-training in order to remain effective. The risk, especially with the latter approach, is that a statistical anomaly-based IDS may unintentionally learn to accept more and more anomalous activity as normal activity (or vice versa) over time, thereby allowing a less detectable avenue of access by intruders. Periodic inspection and validation of a classifier can help avoid classifier creep.

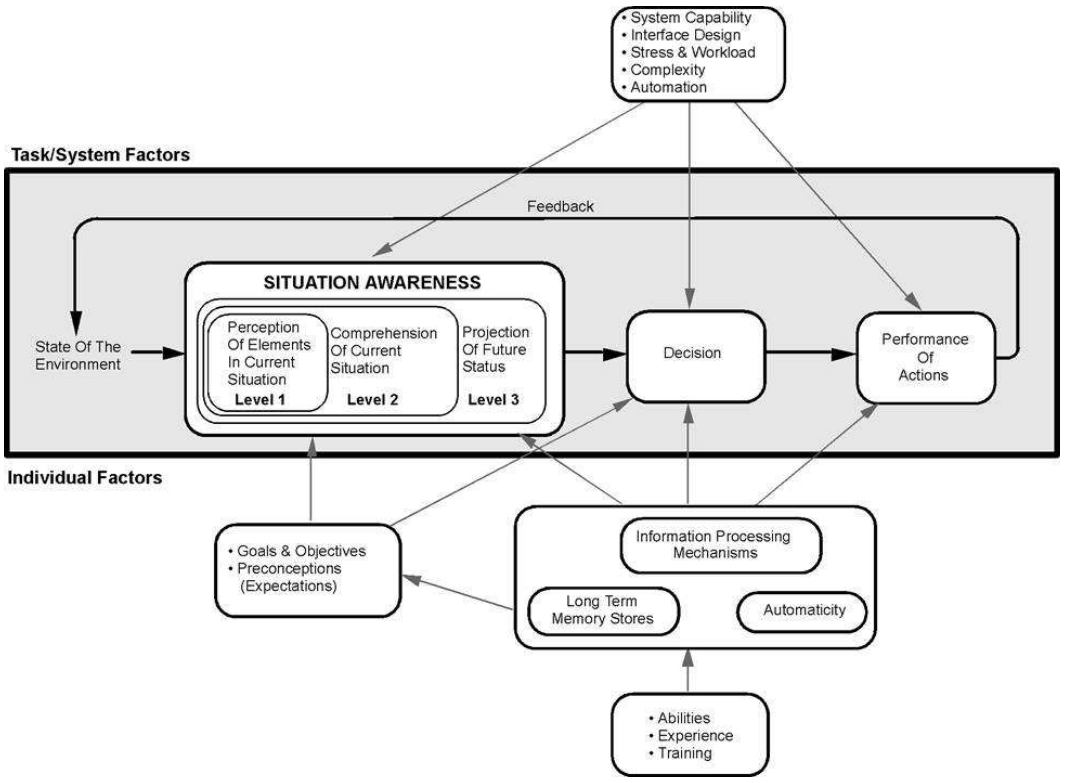


Figure 2.4: Endsley's Situational Awareness Model [32].

2.5 Situational Awareness

While other works [31] [61] define situational awareness concepts, the most pervasive and relevant situational awareness concepts in literature today are proposed by Dr. Mica Endsley [32]. An overview of Endsley's conceptual model of situational awareness, and its applicability to the cyberspace domain, are discussed herein.

2.5.1 Endsley's Model. The dominating definition of Situational Awareness (*SA*) was proposed by Dr. Mica Endsley: "the *perception* of the elements in an environment within a volume of time and space, the *comprehension* of their meaning and the *projection* of their status in the near future" [32]. The process which initiates and refines SA (shown in Figure 2.4) involves sensing the environment, making decisions, performance of actions, and feedback. While SA does not guarantee good decisions, it is a precursor for decision making. SA is attained through various sources of information.

2.5.1.1 Levels of SA. Coinciding with her definition of SA, Endsley defines three levels of SA: perception, comprehension and projection [32]. Each level of SA builds upon the previous.

- Perception is the most basic of the three levels, and is the ability to perceive domain elements within time and space. Endsley [32] states that "limits in perception (or attention to percepts) drastically hinder one's ability to formulate even the most basic understanding of one's environment." Applied to CSA, one wants to be able to perceive cyber events taking place within the system or across the network, e.g. "ip 123.123.123.123 is attempting an ARP poisoning attack against 10.0.0.1." To build this perception may require data from only one sensor or data from a combination of two or more sensors. Without the correct sensors and sensor data, a system will not be able to perceive the event.
- Comprehension builds upon perception; by integrating how people combine, interpret, store and retain information, comprehension provides "operationally relevant meaning" [32]. In the above example, suppose IP 10.0.0.1 was an email server in the internal network. Certainly this knowledge provides a different picture than if the IP belonged to a web server in ourthe network's demilitarized zone (DMZ). If 10.0.0.1 belonged to an email server, it implies that the intruder has been able to bypass one of the network's routers, a firewall and perhaps a proxy server. It implies much deeper penetration than if the server was some publicly accessible asset. It also implies, perhaps, a much more malicious and imminent intent to disrupt, deny, or degrade the network's confidentiality, integrity or availability. It is important to note that specific domain knowledge would be required to draw these conclusions. An expansion of domain knowledge can only help to improve a system's ability to comprehend the environment.
- Projection is Endsley's highest form of SA; it is the ability to "anticipate or forecast future events, allowing for timely decision making" [32]. Continuing the illustration of the ARP poisoning attack on the email server, given that

the current state is perceived and comprehended, S.A. would use projection to anticipate future states. Such an anticipation is necessary for determine that whoever was at 123.123.123.123 was attempting to disrupt communication capabilities as the first stage of a larger attack.

2.5.1.2 Attention. Endsley [33] also discusses the impact of *working memory and attention* on SA. In human factors, a person can only pay attention to so many things simultaneously. The same can be said for computers. Due to time, memory and processing constraints, an SA system cannot always attend to all sensors. Doing so would cause a system to become non-responsive, a self-imposed denial of service. Instead one must pick and choose the most operationally relevant sensors to attend. Distraction comes in the form of overloaded or interrupted processes, or from attending to sensors or sensor data that does not provide as large of a cost benefit ratio. Some sensor costs are processing time, processing power, noise and deficient trust (errors of ommision/commission rates). The benefit is the value of the information, e.g. its trustworthiness, successful perception rate and importance/relevance of the data collected.

2.5.2 Cyberspace Situational Awareness (CSA). SA is applicable to every operational environment (big or small) you can conceive of, and cyberspace is no exception. Cyberspace Situational Awareness (*Cyber SA, or CSA*) is the application of *situational awareness* to develop an understanding of the cyberspace domain. Without some level of CSA, there is no hope of securing networks today and into the future.

According to [39] [15] [66], CSA can be obtained by gathering and correlating data simultaneously from multiple sensor inputs. Each sensor's observations should be compared against other observations along with a knowledge base, capturing the *relevant* details about the event so that it can be analyzed for potential problems, and reporting this information to a data repository. Today, this analysis is often done by a human operator, and is typically a reflexive vice proactive act: to determine "what just happened" vs "what is about to happen". There are simply too many places to look in today's networks to obtain an accurate and timely site picture; there are too many sensors and too many moving parts for a human to hunt them all down. Having a single place to look for sensory output can streamline this process.

Once sensor data outputs such as system audit log entries are gathered into a central repository, they need to be analyzed for potential threats as quickly as possible. Without timely situational awareness, cyberspace security professionals will be unable to adequately protect cyberspace resources. It should be apparent that automation is a key enabler to attaining enhanced CSA.

2.5.2.1 CSA System Requirements. Okolica, et al. [57] propose a framework of three overlapping activities are needed to develop an automated Cyber SA system. First, they propose the system needs to develop a test environment which provides real, timely, functional, scaled and heterogeneous sensor data that can be correlated and fused. Second, the Cyber SA system needs to develop language(s), consisting of cyberspace-relevant vocabulary and grammar, which can describe the cyber environment at different levels of abstraction. Finally, a Cyber SA system needs to integrate the adversarial narrative into the abstraction space.

2.5.2.2 NetD COP. An ongoing initiative at Air Force Research Labs in Rome, NY is the Network Defense Common Operational Picture (NetD COP). According to [23], the NetD Cop effort demonstrates "enhanced situational awareness and visualization techniques for network defense". The system is comprised of an event correlation engine for cyber attack recognition (ECCARS) and a collection of visualization tools (VIAssist, VisAlert and FlexViewer). The conclusion from this demonstration was that the system is capable of providing and enhancing situational awareness on live network discs [23]. One main piece to the NetD Cop is its Data Extraction Utility (DEU), a client/server application which parses sensor information fed to it from firewalls and intrusion detection systems. However, it should be noted that the sensor inputs to the DEU is already processed (not raw) information, an

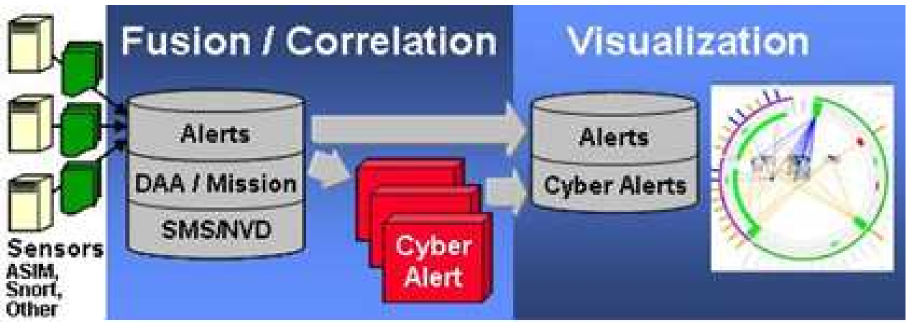


Figure 2.5: AFRL's NetD COP Decision engines create high-level security alerts with mission impact assessment from a variety of sensor and data inputs, using a plug-in architecture [23].

abstraction/classification decision has already been made as to whether or not low level forensic evidence is suspicious or not. Additionally, NetD COP is focused on alerts based on network traffic and not on forensic evidence that may be present on host systems which could indicate attack.

2.5.2.3 SAIL. Another existing architecture is Situational Awareness Inference and Logic (SAIL). This system architecture proposes to provide higher-level reasoning than other SA systems [12]. The model provides functionality in three stages which loosely correlate to Endsley's SA model. According to the author, all other known SA systems developed (through publishing in 2009) leave too much of the situation assessment and projection still with the human observer.

Largely based on formal logic, SAIL provides a controlled natural language (CNL), designed to reduce ambiguity/vagueness) interface for accepting both data streams and human inputs for sensor data. Figure 2.6 illustrates the SAIL architecture, and is outlined as follows. Sensor data is collected and aggregated (summarized) like other SA systems. The aggregate data is then fed to a semantic analysis layer, which applies descriptive logics to map the data points through an ontology in order to build a situation assessment. The act portion of Endsley's SA model is performed by SAIL's alerter. As the name implies, the alerter notifies personnel of critical issues it has reasoned about. (SAIL provides a manual query function as well.)

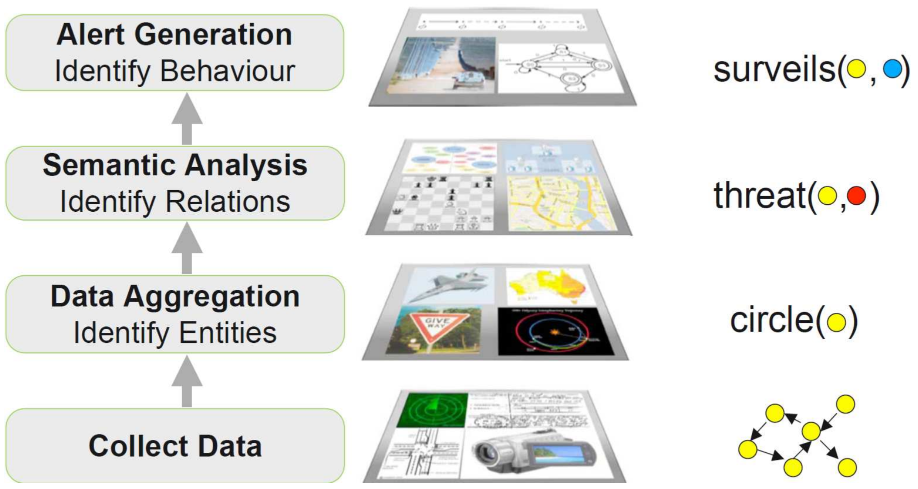


Figure 2.6: Baader's SAIL system architecture [12].

While SAIL may be potentially adapted for use as an IDS reasoner, it has not been applied to the cyberspace domain. SAIL relies upon a human operator to feed it relevant features by which it can make inferences about the domain it monitors. Although SAIL performs data aggregation and semantic analysis from raw data to build inferences, it not does not use specifically apply CRISP-DM's process to do so.

2.5.3 JDL Data Fusion Model. Another existing situational awareness model is JDL's data fusion model. The JDL's data fusion model is maintained by the JDL Data Fusion Group, and is reportedly [65] the most widely-used method for categorizing data fusion-related functions. The JDL model provides a more bottom-up data-centric approach than Endsley's model. The JDL Data Fusion Model defines five data fusion levels, as depicted in Figure 2.7.

- Level 0 - Sub-Object Data Assessment: estimation and prediction of signal/object observable states on the basis of pixel/signal level data association and characterization;
- Level 1 - Object Assessment: estimation and prediction of entity states on the basis of observation-to-track association, continuous state estimation (e.g. kinematics) and discrete state estimation (e.g. target type and ID);

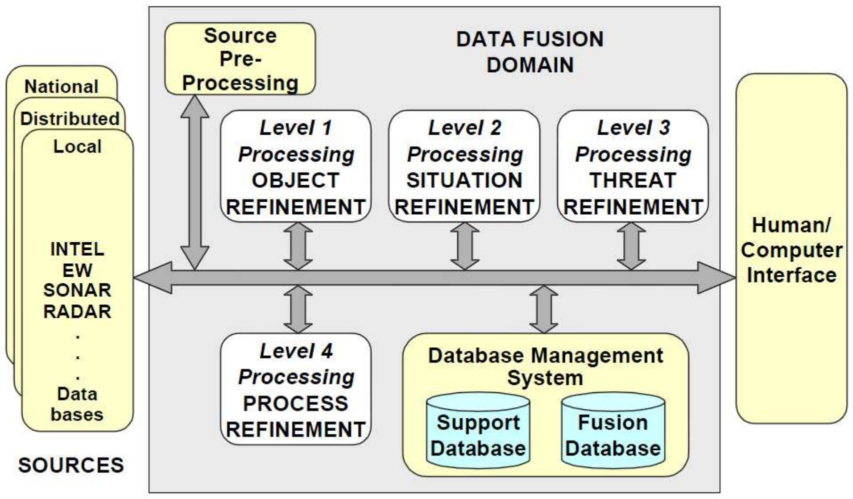

National
Distributed
Local
INTEL
EW
SONAR
RADAR
.
.
.
Data bases
SOURCES
Source Pre-Processing
DATA FUSION DOMAIN
Level 1 Processing OBJECT REFINEMENT
Level 2 Processing SITUATION REFINEMENT
Level 3 Processing THREAT REFINEMENT
Level 4 Processing PROCESS REFINEMENT
Database Management System
Support Database
Fusion Database
Human/ Computer Interface

Figure 2.7: Data Fusion Group's Joint Data Library data fusion model for situational awareness [65].

- Level 2 - Situation Assessment: estimation and prediction of relations among entities, to include force structure and cross force relations, communications and perceptual influences, physical context, etc.;
- Level 3 - Impact Assessment: estimation and prediction of effects on situations of planned or estimated/predicted actions by the participants; to include interactions between action plans of multiple players (e.g. assessing susceptibilities and vulnerabilities to estimated/predicted threat actions given ones own planned actions);
- Level 4 - Process Refinement (an element of Resource Management): adaptive data acquisition and processing to support mission objectives.

2.5.4 INFERD. INformation Fusion Engine for Real-time Decision-making (INFERD) [66] is another context-aware framework which has been applied to the IDS problem. INFERD utilizes to the JDL model of as a measure of how well it provides SA. INFERD is, in a nutshell, an alert correlation engine, which does not look at low level host-based forensic evidence, rather, it relies upon a series of host-based network sensors to decide between normal and malicious events. Stotz [66] makes no mention

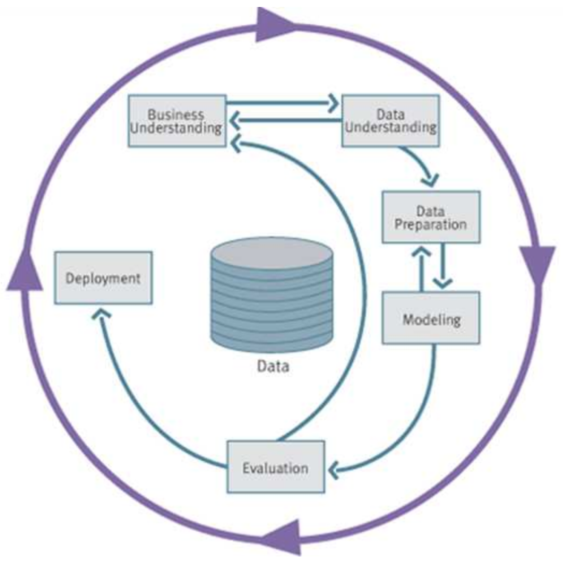


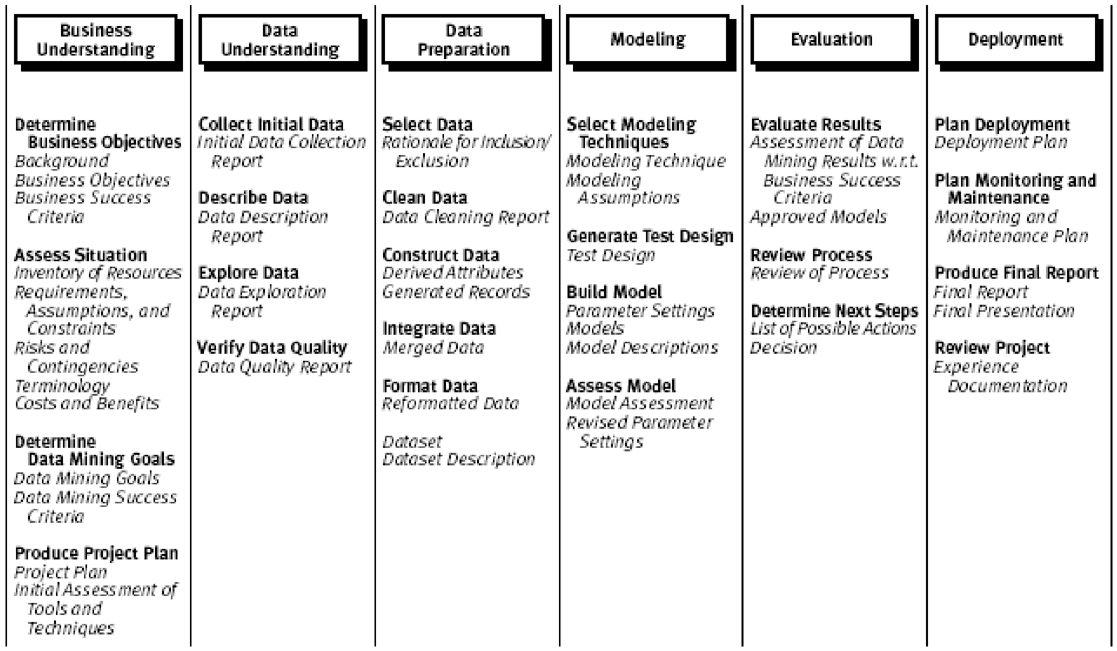
Figure 2.8: The Cross-Industry Standard Process for Data Mining is a cyclic process consisting of six phases [21].

of attending to host based sensors which monitor process metadata or internal system events such as file I/O.

2.6 *Knowledge Discovery and Data Mining*

The goal of knowledge discovery and data mining (KDD) is to extract patterns from data. There are a number of professional bodies dedicating to furthering data mining research, some prominent ones are the Association for Computing Machinery (ACM) Special Interest Group for Knowledge Discovery and Data Mining (SIGKDD) and the Data Mining Group (DMG). The data mining concept is nothing new, people have been doing it manually for centuries. However, as problems become more complex and require more data, there is an increased need for digital processing to mine data. A repeatable data mining process was established by the pioneers of the data mining movement [21] to provide guidelines for how a data mining process should flow, and is discussed in the next section.

2.6.1 CRISP-DM. The CRoss Industry Standard Process for Data Mining (CRISP-DM) [21] outlines a six-phase cycle for data mining projects, as shown



Business Understanding	Data Understanding	Data Preparation	Modeling	Evaluation	Deployment
Determine Business Objectives *Background* *Business Objectives* *Business Success Criteria*	**Collect Initial Data** *Initial Data Collection Report*	**Select Data** *Rationale for Inclusion/ Exclusion*	**Select Modeling Techniques** *Modeling Technique* *Modeling Assumptions*	**Evaluate Results** *Assessment of Data Mining Results w.r.t. Business Success Criteria* *Approved Models*	**Plan Deployment** *Deployment Plan*
Assess Situation *Inventory of Resources* *Requirements, Assumptions, and Constraints* *Risks and Contingencies* *Terminology* *Costs and Benefits*	**Describe Data** *Data Description Report*	**Clean Data** *Data Cleaning Report*	**Generate Test Design** *Test Design*	**Review Process** *Review of Process*	**Plan Monitoring and Maintenance** *Monitoring and Maintenance Plan*
Determine Data Mining Goals *Data Mining Goals* *Data Mining Success Criteria*	**Explore Data** *Data Exploration Report*	**Construct Data** *Derived Attributes* *Generated Records*	**Build Model** *Parameter Settings* *Models* *Model Descriptions*	**Determine Next Steps** *List of Possible Actions* *Decision*	**Produce Final Report** *Final Report* *Final Presentation*
Produce Project Plan *Project Plan* *Initial Assessment of Tools and Techniques*	**Verify Data Quality** *Data Quality Report*	**Integrate Data** *Merged Data*	**Assess Model** *Model Assessment* *Revised Parameter Settings*		**Review Project** *Experience Documentation*
		Format Data *Reformatted Data*			
		Dataset *Dataset Description*			

Figure 2.9: Generic tasks of the CRISP-DM Phases. The Data Understanding phase consists of collecting initial data, describing the data, exploring the data and verifying data quality [21].

in Figure 2.8. CRISP-DM is intended to be industry-independent, tool-independent and application-independent. The goal of CRISP-DM is to provide organizations an understanding of the data mining process and provide a road map to follow while planning and carrying out a data mining project. The phases are business understanding, data understanding, data preparation, modeling, evaluation and deployment. The second phase, data understanding, "starts with initial data collection and proceeds with activities in order to get familiar with the data, to identify data quality problems, discover first insights into the data or to detect interesting subsets to form hypotheses for hidden information" [21]. The hierarchical methodology defines each phase as a set of generic tasks each consisting of a set of specialized tasks and finally process instances. The generic tasks which make up data understanding are shown in Figure 2.8.

2.6.2 KDD Platforms. There are plenty of tools available for data mining, ranging from open and free software to enterprise solutions costing hundreds of thousands of dollars. A small sampling includes WEKA, RapidMiner, JDM and MATLAB. First, Waikato Environment for Knowledge Analysis (WEKA)is an open source Java project hosted by the University of Waikato, New Zealand. WEKA provide a graphical user interface called Explorer, which allows for interactive analysis of datasets by building workflows, and the ability to build and test computer models. WEKA has an API allowing for integration into developers' Java programs as well.

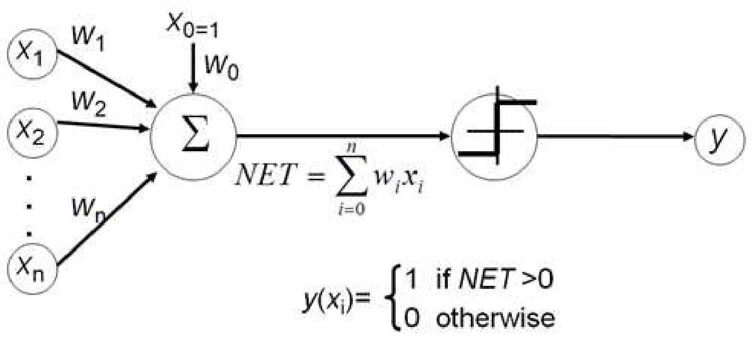
Second and formally known as YALE, RapidMiner boasts more than 400 data mining operators, and a "huge amount of visualization techniques (http://rapid-i.com/content/blogcategory/38/69/). While RapidMiner is commercial software, it offers a community edition and a Java API.

The third KDD platform Java Data Mining API is an effort to divorce the technology from any particular vendor. However, it is not as mature as WEKA or

Finally, used in many universities and research laboratories, MATLAB ("Matrix Laboratory") is a commercial product developed in C and Java. MATLAB is a powerful matrix manipulation tool which can be used for regression and classification. MATLAB provides extensions, called toolboxes, allowing for specialized support. MATLAB's neural network toolbox is used to test the performance of selected features.

2.6.3 KDD Algorithms. Data mining algorithms apply machine learning to extract patterns from observed data. There are three main forms of machine learning: supervised, unsupervised and reinforcement learning. Each technique has its pros/cons. The most direct form of machine learning is *supervised learning* [70].

Supervised learning involves learning a function from examples of its inputs and outputs. Typically with supervised learning, an expert provides class labels to each observation in the dataset, and a machine is then trained to recognize each class based on its labels and prototypical feature values. This process is known also as classification.



x_1
w_1
$x_{0=1}$
w_0
x_2
w_2
Σ
x_n
w_n
$NET = \sum_{i=0}^{n} w_i x_i$
y
$y(x_i) = \begin{cases} 1 & \text{if } NET > 0 \\ 0 & \text{otherwise} \end{cases}$

Figure 2.10: The perceptron is the basic unit of decision within a neural network [35].

The next form of machine learning is *unsupervised learning.* Unsupervised learning involves learning patterns in the input when no specific output values (labels) are supplied. A machine using this technique attempts to group data to

The final form of machine learning is *reinforcement learning.* Reinforcement learning methods are based on iterative feedback which is used to maximize matching. Effective reinforcement learning techniques balance exploration and exploitation; poor results are obtained if too much emphasis is placed one one or the other.

2.6.4 Artificial Neural Networks. Also known as multi-layer perceptrons, there are a number of artificial neural network (ANN) algorithms used for machine learning, including associative memory, feed forward and back propagation ANN [40].

The simplest form of an ANN algorithm is a single perceptron, as shown in Figure 2.10 [40]. Perceptrons, and all ANNs for that matter, are biologically inspired from the way the human brain works. When a perceptron is provided enough "energy" through weighted inputs to meet the minimum threshold as established by the sigmoid function (depicted downstream from the summation before arriving at the next perceptron "y" in Figure 2.10), it "fires", sending a 1 down the wire to the next perceptron. Otherwise, it does not fire; it sends a 0 down the wire. Synaptic weights $\omega_{0..n}$ are used to imply relationships between inputs and the perceptron itself; the higher the weight, the stronger the bond between the input and the perceptron.

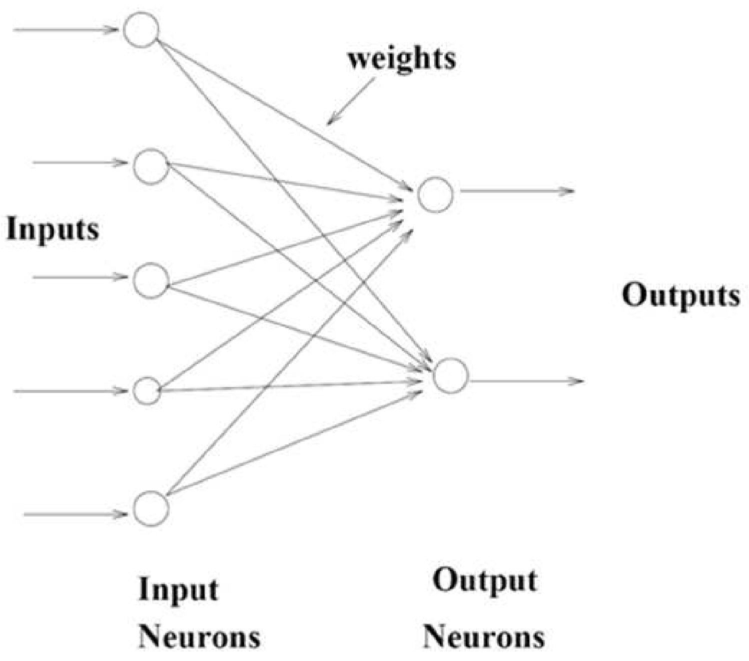


Figure 2.11: An associative memory neural network is an ANN with exactly two layers of neurons and two connecting layers of weights [35]

.

Weights are usually between 0 and 1, but can be assigned negative values for implying inverse relationships between an input and the perceptron.

The purpose of an *associative memory* ANN (Figure 2.11) is, as the name implies, to learn associations [40]. This network consists of two layers: an input layer and an output layer. The input layer receives weighted inputs from the environment; each input layer perceptron will either fire to signal each of the output layer perceptrons or it won't, again based on the sigmoid function. An associative memory ANN is iteratively taught to recognize a particular pattern, and once trained, to recall the pattern even when presented with incomplete or noisy data.

Multi-layer ANNs include one or more "hidden" (intermediate) layers of perceptrons. These networks are capable of learning non-linear relationships, which are likely to occur in a high-dimensional or complex space. A popular method for training multilayer perceptrons is through *back propagation*. With back propagation, a network is trained in two repeated phases: the forward phase and the backward phase.

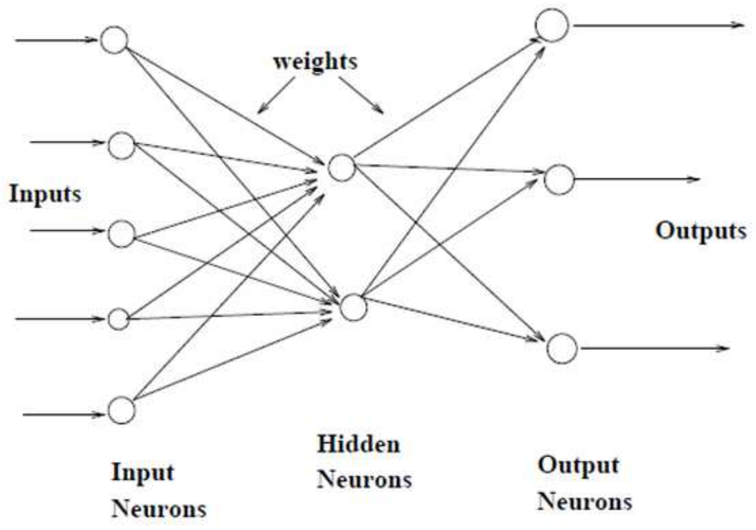


Figure 2.12: An ANN consisting of three layers of neurons and two connecting layers of weights. The middle layer is hidden as it is neither input or output [35].

In the forward phase, the signal is fed from each of the inputs straight through to the output, without any adjustment of the synaptic weights. In the backward phase, the output signal is compared to the desired output, and the error is propagated backwards through the ANN so that the synaptic weights can be fine-tuned [40]. Together, the two phases make up a single epoch. Epochs are used to iteratively update the performance of the neural network until it converges to a minimum mean squared error.

2.6.5 Analyzing results. There are a number of ways to analyze the expected performance of a classifier, prominent methods are through generating a performance plot, a confusion matrix, or a receiver operating characteristic (ROC) curve.

2.6.5.1 Performance Plot. The performance plot provided by Matlab's Neural Network Toolkit [29], displayed in Figure 2.13, visualizes the mean squared error of the neural network network, which indicates the expected accuracy the network will have when classifying data. Initially, a network starts with a high MSE, but as the network is trained, the MSE decreases (the network learns). The performance plot consists of three lines, representing the mean squared error of three subsets of

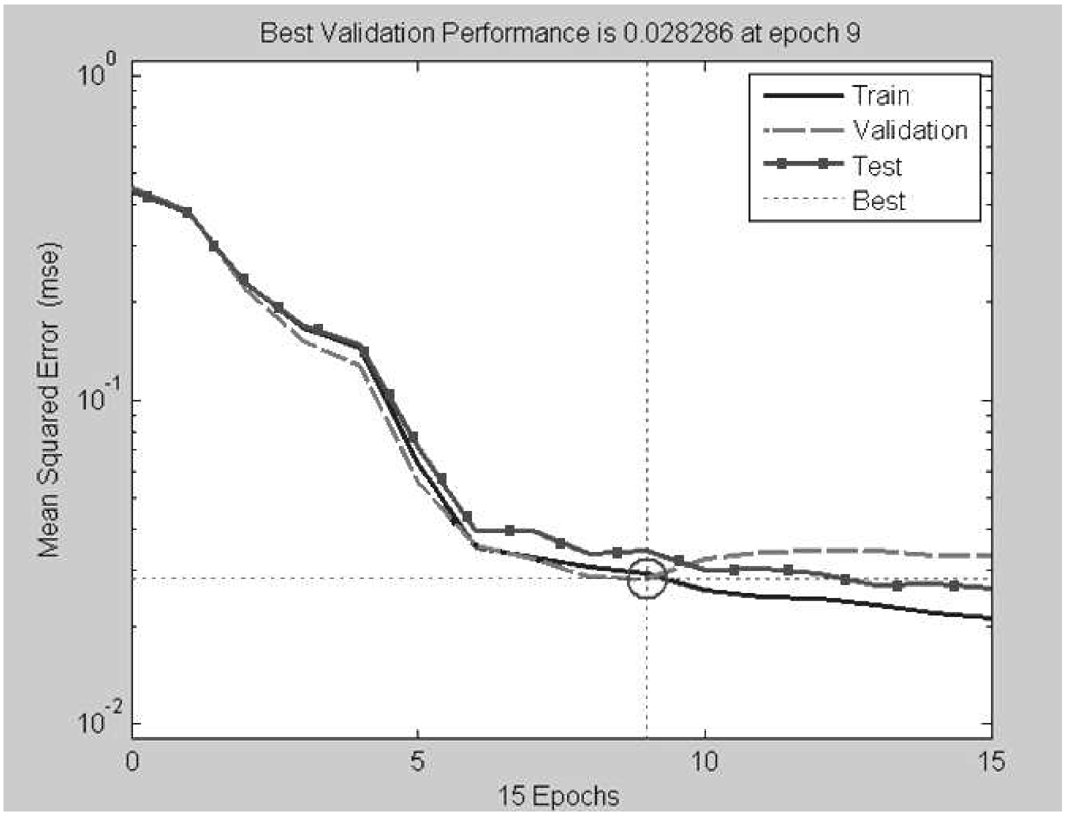


Figure 2.13: Example of a performance plot, used to analyze the mean-squared error of the classifier. A smaller mean-squared error indicates higher performance. This plot indicates that the best performance (lowest MSE) for the neural network was attained after nine training epochs.

Table 2.3: A confusion matrix is used to analyze the overall success rate of a classifier by accounting for errors of omission and errors of commission.

Output Class				
1	771 (82.2%)	118 (12.7%)	11 (1.2%)	85.7% (**14.3%**)
2	0 (0.0%)	26 (2.8%)	0 (0.0%)	100% (**0.0%**)
3	0 (0.0%)	0 (0.0%)	5 (0.5%)	100% (**0.0%**)
	100% (**0.0%**)	18.1% (**81.9%**)	31.3% (**68.8%**)	86.1% (**13.9%**)
	1	2	3	
	Target Class			

vectors from within training population provided: training vectors, validation vectors, and test vectors, in a 60%-20%-20% split respectively. Training continues the network is memorized or some minimum mean squared error requirement is met, which helps to avoid overfitting /biasing the classifier to the specific data provided.

A confusion matrix, as shown in Table 2.3, illustrates a classifier's accuracy rates by tabulating correct and incorrect results. The 3x3 grid in the upper left of the matrix indicates for each class, which items were classified correctly into the class (the light gray cells), and which items were classified incorrectly (the medium gray cells). Each of the cells (or bins) in this 3x3 grid include two numbers: the raw count of observations which are in the bin, and its corresponding percentage out of the total population. The column on the far right indicates, for each output class, the correct commission rate, along with the error of commission rate in bold. The row at the bottom indicates, for each target class, overall successful commission rate, along with errors of omission. In Table 2.3, there were 118 mislabeled "Class 2" observations, which contributed to output class 1's 14.3% error of commission rate, and to the target "class 2" 81.9% error of omission rate.

The target class is the true label of the data, whereas the output class is what the classifier labeled the data. The right column and bottom row hold summary data

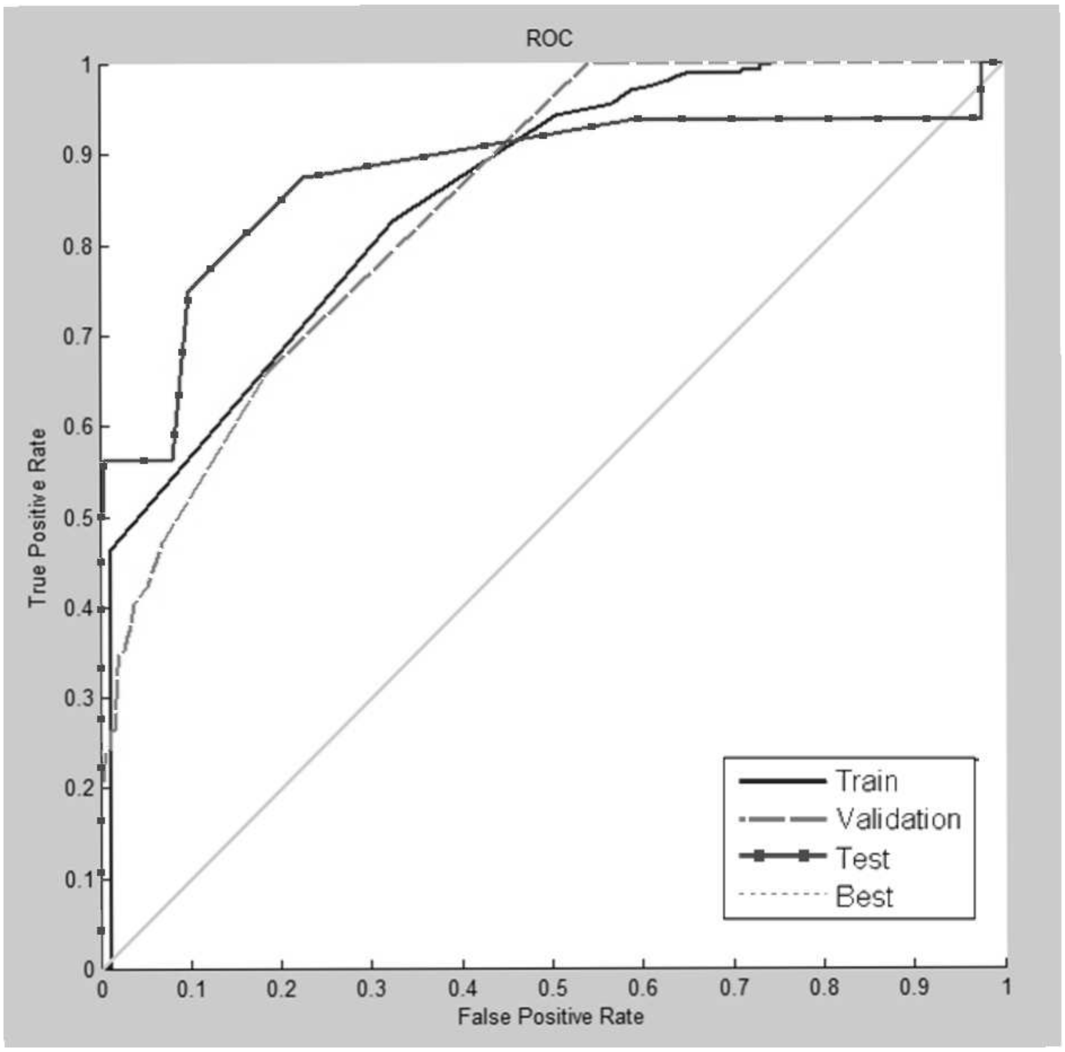


Figure 2.14: Example of a ROC curve, used to analyze a classifier's specificity versus its sensitivity [29].

from the 3x3 grid, and the bottom right corner presents the overall accuracy of the classifier. In this case, the classifier is expected to correctly identify 86.1% of future observations, assuming the data which trained the network was not biased.

2.6.5.2 ROC Curves. The Receiver Operating Characteristic curve (ROC) plot [29], illustrated in Figure 2.14 is a visualization of a classifier's true positive ratio (correct classification rate) charted against its false positive ratio (errors of commission). A ROC curve demonstrates the tradeoff between how sensitive a network is and how fully specified it is. A more accurate ROC curve more tightly fits against the left and top edges of the chart; a less accurate test is indicated by a curve which approaches the 45 degree line of the chart.

2.7 Data Sources

Due to privacy concerns, and to allow for comparable results, many researchers use test data sets rather than using real, operational data. There are two main data sets researchers in intrusion detection use, as well as test environments for those who want to generate new data sets for their experiments.

2.8 Research Data Sets

Much of intrusion detection research community relies heavily upon comparing results based on datasets from two research groups. Both data sets are network packet captures taken during periods of normal activity and network attacks.

The first dataset was a set of tcpdump files created in 1998 (additional iterations were created in 1999 and 2000) by a partnership between Massachusetts Institute of Technology's (MIT Lincoln Laboratory and the Defense Advanced Research Projects Agency DARPA), called the "MIT/DARPA Intrusion Detection System Evaluation datasets" or simply MIT/DARPA datasets. Both the MIT/DARPA 1998 and MIT/-DARPA 1999 data sets have since been criticized for poor methodologies which biased results [54], so much so that calls have gone out for researchers to stop using them as benchmarks for IDS research [19].

The second widely-used IDS datasets were benchmarked by the KDD Cup in 1999. Unfortunately, as this data is an extracted subset of the MIT/DARPA '98; it faces the same scrutiny and pitfalls [18].

Because existing datasets do not provide host-based sensor data, and the desire to have event-correlated network and host sensor data, a separate data set will need to be generated to support the research performed for this thesis.

2.9 Summary

A literature review of many facets related to this thesis research includes the study of intrusion detection systems and techniques, situational awareness and its applicability to the cyberspace domain, along with knowledge discovery and neural network methodologies. Upon a literature review of leading intrusion detection efforts, it appears that a multi-sensor study of a system under attack has not been attempted or documented for the purpose of performing host-based forensic feature selection. This thesis applies knowledge discovery techniques specified by CRISP-DM to identify relevant intrusion detection features, and validates these results by testing and analyzing the performance of these features with an artificial neural network.

III. Methodology

In order to identify the forensic data which best indicates various stages of an attack on a host, a series of controlled data collections are performed. The data is then explored and analyzed for consistency and usability for identifying an event from the collection via the CRISP-DM process illustrated in Figure 3.1. Selected features are used to build a neural network to test the performance of the selected items.

This chapter outlines the research methodology used to study the ability of combining behavior-based classifiers for detecting network attacks. First, is a description of the data collection environment, followed by a explanation of how CRISP-DM's data understanding, preparation, modeling and evaluation steps are used to analyze the forensic features for their relevance in detecting malicious host activity. Results of this process are presented in Chapter four.

3.1 CRISP-DM: Data Understanding

The data collection environment, the "Cyberspace Situational Awareness Laboratory" (*CSA Lab*), consists of a "black hat" computer and a "target" machine hosted on a wireless network, as depicted in Figure 3.2. The target machine is a Windows XP-based virtual machine which is outfitted with a selection of live forensic data capture tools. The black hat machine is a Windows XP system (not a virtual machine), outfitted with a selection of well known scanning and exploit programs. Both machines are Dell Latitude$^{\text{TM}}$ D630 laptops: a 64-bit hardware architecture with an Intel® Core$^{\text{TM}}$ 2 Duo processor, 8GB of RAM, a 120GB hard drive and a wireless network interface card (NIC). The host network provides all routing, domain name resolution and Internet access. A specification of the relevant hardware and software used is listed within this section.

3.1.1 Black Hat Client. The black hat machine is a workstation outfitted with a number of cyber attack tools. During data collections, the black hat client

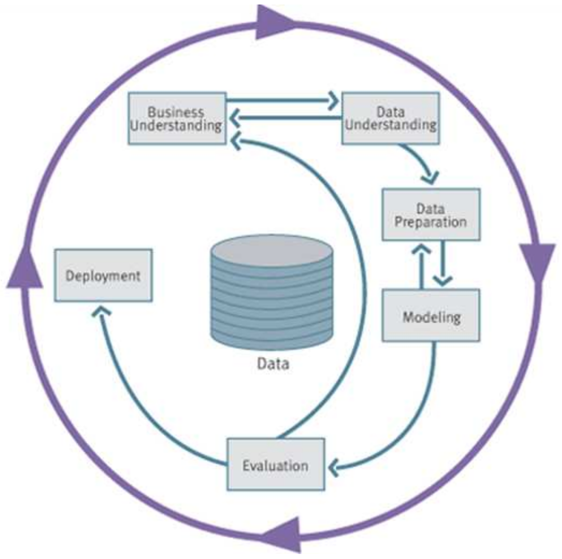


Figure 3.1: Data Understanding is one of the key phases of the CRISP-DM Process Model [21].

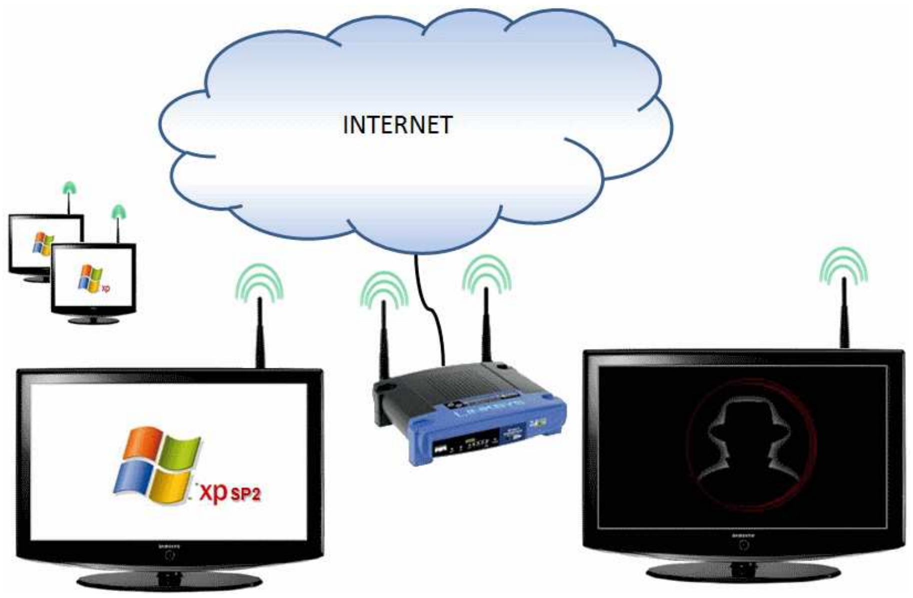


Figure 3.2: The CSA Lab environment is comprised of two machines connected to a wireless network, with Internet connectivity provided by the host network.

Table 3.1: The relevant software configuration for the black hat machine includes a number of well-known scanning and exploit tools.

Machine Name	BLACKHAT
Host Operating System	Windows XP Service Pack 2
Applications	Microsoft Office 2007 Professional, Microsoft Internet Explorer 7.0
Scanning & Exploit Tools	ping: Included with MS Windows XP, SP2
	fping v2.2: www.kwakkelflap.com
	ws_ping v2.30: www.ipswitch.com
	Zenmap v4.76: www.nmap.org
	Nessus v4: www.nessus.org
	DumpSec v2.8.6: www.systemtools.com
	ShareEnum v1.6: www.sysinternals.com
	MetaSploit v3: www.metasploit.com
	PWDump v2.0.0-beta-2: www.foofus.net
	FGDump v2.1.0: www.foofus.net
	RegBack: www.microsoft.com
	BO2K v1.1: www.bo2k.com

runs "cyber attack" scenarios as described in Section 3.1.2 and detailed in Appendix C. The configuration of the black hat client is shown in Table 3.1.

3.1.1.1 Target Machine. The machine which performs the data collections (referred to as the "target") is a virtual machine instance of Windows XP, Service Pack 2, running on top of a 64-bit Microsoft Windows 7 Enterprise host operating system. VMWare Workstation 6.05 build-109488 provides the virtualization support, which provides the capability of create an image of the machine's state, as well as the ability to quickly restore the machine's state to the same baseline between data collections quickly.

The target is outfitted with live forensic tools which monitor network, process and file activity. During experiments, the target runs through scripted "Normal user" activity scenarios as specified in Appendix B. Additionally, the target is subjected to attack activity, using the scenarios as specified in Appendix C. Table 3.2 outlines the software installed on the target.

Table 3.2: Configuration of the "target" machine. Sensors are annotated as either transactional (t) or snapshot (s) sensors, as explained in Section 3.1.2.

VM Name	TARGET
VM Configuration	2GHz Intel Core 2 Duo CPU, 2GB RAM, 30GB hard drive, bridged ethernet adapter
Guest OS	Windows XP Service Pack 2
Applications	Microsoft Office 2007 Professional, Microsoft Internet Explorer 7.0
Sensors	[t] logman.exe v5.1.2600.2180: Included with MS Windows XP, SP2
	[t] tshark.exe v1.2.4: Included with Wireshark v1.2.4 [4]
	[s] listdlls.exe v2.25: www.sysinternals.com [59]
	[s] logonsession.exe v1.1: www.sysinternals.com [59]
	[s] pslist.exe v1.28: www.sysinternals.com [59]
	[s] tcpvcon.exe v2.54: www.sysinternals.com [59]

3.1.2 Sensors. To capture forensic evidence on the fly, two types of sensors are used: transactional sensors and snapshot sensors. As the name implies, transactional sensors capture each transaction within the sensor's purview. Transactional sensors, once started, continue capturing data until they are halted, or until some user-defined rule is met (such as time limit or file size thresholds). An example of this type of sensors is tshark.exe [4], used to monitor a network device for inbound and outbound network packets. Snapshot sensors capture, and provide raw and/or summary data about a portion of the current system state or configuration. An example of a snapshot sensor is SysInternal's pslist.exe [59], which displays a snapshot of process memory and processor utilization. System events which start and stop between snapshots may yield no forensic evidence to a snapshot sensor of any sort.

Collection is performed in three stages: *normal*, *scanning*, and *exploit* collections. This helps to keep collection sizes manageable, and to provide distinct datasets based on the purpose for the collection. The first steps to each collection involves ensuring the target machine is set to its baseline configuration by loading a previously established VM snapshot through the VMWare software, and launching the sensor scripts. To launch sensors for each experiment in a repeatable and consistent manner, Windows Script Host (WSH version 5.6), paired with DOS batch programming, is used. A source code listing and a brief explanation of the sensor scripts are listed in Appendix A.

The purpose of the *normal* collection is to generate a set of data which consists only of typical user activity, devoid of scanning or other malicious activity. The normal collection only involves performing actions on the target machine. During this collection, summarized in Appendix B, the target machine runs through a series of scenarios such as using Microsoft Office products, Internet Explorer and sending and receiving email. The data set generated is used to provide a baseline forensic data set to compare with the scanning and exploit collections.

The purpose of the *scanning* collection is is to generate a set of data which consists of scanning and service enumeration events. The scanning collection involves both machines, but the bulk of the activity is performed by the black hat machine. As with other collections, the target machine launches the forensic sensors to start the collection process. Then the black hat machine is used to run through a series of scanning enumeration activities against the target machine, such as using ping, Nessus and Nmap. The data set generated is used to provide a better understanding of forensic evidence generated by a scanning event versus normal activity.

The purpose of the *exploit* collection is is to generate a set of data which consists of attempts to gain access, elevate privileges, plant back doors and to cover tracks. The scanning collection involves both machines, but the bulk of the activity is performed by the black hat machine. As with other collections, the target machine launches the forensic sensors to start the collection process. The black hat machine is used to run through a series of exploit activities against the target machine, such as using Metasploit to launch various exploit payloads at the target machine, using regback, pwdump and fgdump, and other tools to exfiltrate data, and using back orafice to launch a keylogger. The data set generated is used to provide a better understanding of forensic evidence generated by a various exploitation stages within an attack sequence.

Each dataset consists of outputs from the six sensors listed in Section 3.1.1.1, with potentially thousands of files to parse through. Each tool generates specific forensic data features. Each feature generated is initially considered a candidate for providing separation between normal and malicious activity, in order to identify as many effective features as possible.

Process Overview Sensor. In order to capture summary process data, such as memory utilization, user and system time, number of threads and handles, SysInternals' pslist.exe (version 1.28) [59] program is used. Running this tool from the command line with the $-x$ parameter provides process ID, process name, user and kernel time, counts of threads, handles, context switches and various memory statistics. This tool is categorized as a snapshot sensor. By adding $-s - r2$ parameters, a snapshot is obtained every two seconds; this data is streamed directly to an output file from the console. While pslist.exe is present in digital forensics research [9] [10] [68], no literature has presented itself which persists tool outputs to a central data repository in order to farm for features to use in classification. Each feature produced by the tool is therefore considered a candidate feature for identifying malicious activity.

Table 3.3: "Process Overview" sensor features which are evaluated for effectiveness in distinguishing between normal and attack events.

"Process Overview" Sensor Features evaluated	
Field Name	**Description**
priority	Lowest priority thread for PID
threadCount	Count of thread for PID
handleCount	# File handles for PID
cpuTime	CPU time for process
workingSet	Working set memory used by process
vMem	Virtual Memory (VM) for PID
privateVMem	Private VM used by PID
privateVMemPeak	Private VM Peak for PID
faults	Page Faults encountered by PID
nonpagedPool	Nonpaged Pool cache size
pagedPool	Paged Pool cache size
Δ priority	Since previously polled
Δ threadCount	Since previously polled
Δ handleCount	Since previously polled

Continued on Next Page...

Table 3.3 – Continued	
Field Name	**Description**
Δ cpuTime	Since previously polled
Δ workingSet	Since previously polled
Δ vMem	Since previously polled
Δ privateVMem	Since previously polled
Δ privateVMemPeak	Since previously polled
Δ faults	Since previously polled
Δ nonpagedPool	Since previously polled
Δ pagedPool	Since previously polled

Network Packet Sensor. To capture network packets being sent to and from the host system, the console version of WireShark, tShark.exe (version 1.2.4) [4] is used. TShark was chosen over its GUI (GUI) counterpart (wireshark.exe), due to the fact that it provides the same data with a lighter impact to system resources. This tool is categorized as a transactional sensor.

Since the aim is to identify host-based intrusion detection-relevant forensic data, only packets destined for the target machine - packets sent specifically to the host machine plus broadcasts to the machine's subnet - are monitored. The protocols specifically evaluated are TCP, UDP, ICMP, ARP traffic. Relevant fields captured include, but are not limited to those listed in Table 3.4. There are a number of digital forensics research efforts which utilize packet capture files [38] [36] [46]. The features and methodologies vary from signature detection to anomaly detection, but Haag [38] and Gonzalez [36] both focus on the elements found within protocol headers and packet-level metrics, which is where this research effort spends some of its focus with regard to network traffic features. Additionally, this research looks at volumetric data such as number of bytes sent/received per protocol.

Table 3.4: Some of the “Network packet sensor” features which are evaluated for effectiveness in distinguishing between normal and attack events.

“Network Packet Sensor” Features evaluated	
Field Name	**Description**
Inter-Arrival Time	Average Time between packets
Window Size	Average From TCP header
Payload Size	Average in bytes
# Protocol Packets	Count packets by protocol
# Local Ports	Count distinct ports with communication
# Packets	Count count of packets
# Flags	Counts TCP Flags, UDP/ICMP/ARP Codes

Process Dlls Sensor. To help with the mapping of processes, and which dynamic-linked libraries (DLL)they are associated with, SysInternals' listdlls.exe (version 2.25) [59] program is used. This tool is categorized as a snapshot sensor. While listdlls.exe is present in digital forensics research [68], no literature has presented itself which persists tool outputs to a central data repository in order to farm for features to use in classification. Features in Table 3.5 were selected based on intuition that they may prove good indicators for identifying malicious activity. Features related “abnormal dll” rely upon comparisons with a baseline “normal” activity capture.

Table 3.5: "Process Dlls" sensor features which are evaluated for effectiveness in distinguishing between normal and attack events.

"Process Dlls Sensor" Features evaluated	
Field Name	**Description**
#DLLs	Count of DLLs loaded
Δ DLL Count	Count difference since last polled
Checksum changed	Boolean, change detected since last polled
Abnormal DLL Count	Based on normal activity
Abnormal DLL Count Delta	Based on normal activity

Process to Ports Sensor. To help with the mapping of data between network captures and event trace logs, SysInternals' tcpvcon.exe (version 2.54) [59] program is used. This command line tool provides a listing of ports and their associated processes. This tool is categorized as a snapshot sensor. While tcpvcon.exe is present in digital forensics research [20] [10], no literature has presented itself which persists tool outputs to a central data repository in order to farm for features to use in classification. Features were selected based on a suspicion that they may be good indicators for identifying malicious activity. Features related "abnormal port" rely upon comparisons with a baseline "normal" activity capture. Table 3.6 lists features evaluated for effectiveness in distinguishing between normal and attack events.

Table 3.6: "Process to Ports" sensor features which are evaluated for effectiveness in distinguishing between normal and attack events.

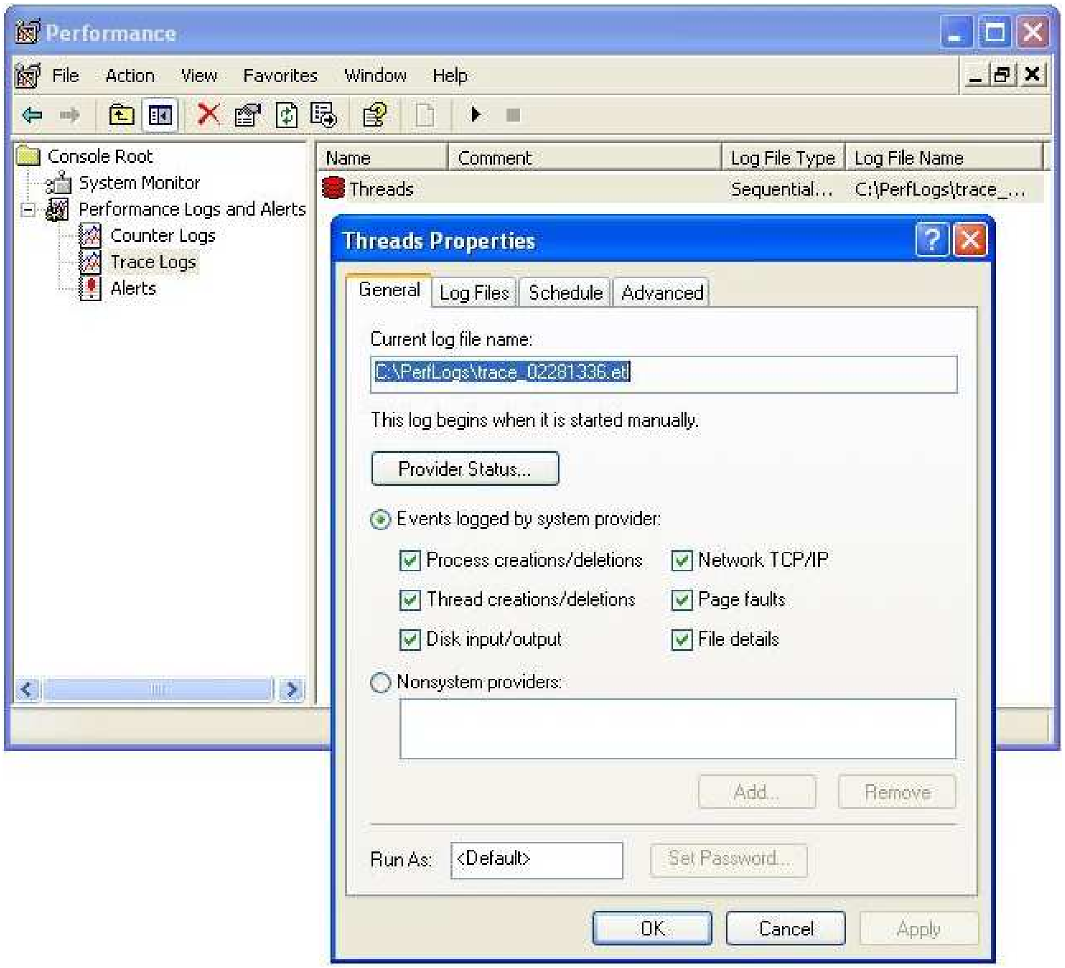
"Process to Ports" Sensor Features evaluated	
Field Name	**Description**
Local Port	Local port associated with PID
Protocol	Local port associated with PID
Remote Port	Local port associated with PID
Local Addr is FQDN	Boolean, Local address specified as workgroup.machine
Local Port Recognized	Based on Windows XP services file
Remote Port Recognized	Based on Windows XP services file
Remote Addr is Self	Remote address is self
State	Boolean sparse matrix: Close_wait, established, listening
State Changed	State changed since last polled
Abnormal Port for Process	Boolean, based on normal activity

Process to Session Sensor. To capture who is logged on at a point in time and what processes a user "owns", SysInternals' logonsessions.exe (version 1.1) [59] program is used. This tool is categorized as a snapshot sensor. While logonsessions.exe is present in digital forensics research [50], no literature has presented itself which persists tool outputs to a central data repository in order to farm for features to use in classification. Features were selected based on a suspicion that they may be good indicators for identifying malicious activity. Features related "abnormal port" rely upon comparisons with a baseline "normal" activity capture. Table 3.7 lists features evaluated for effectiveness in distinguishing between normal and attack events.

Table 3.7: "Process to session" features which are evaluated for effectiveness in distinguishing between normal and attack events.

"Process to session" Features evaluated	
Field Name	**Description**
# Processes	Number of processes tied to session
Δ # Processes	Change since last polled
Logon Type	Boolean Sparse Matrix: Interactive, service, etc
Authentication Package	Boolean Sparse Matrix: NTLM, negotiate
Account Type	Boolean Sparse Matrix: System, local, anonymous, etc

Event Trace for Windows Sensor. In order to capture events each process handles during experiments, Event Trace for Windows (ETW) is used to generate event trace log (ETL) files. This transactional sensor can be configured via Performance Monitor, using the default system provider as shown in Figure 3.3. To start and halt a tracelog capture, the logman.exe (version 5.1.2600.2180) command line utility which ships with the Windows XP operating system is used. While event trace for windows is present in digital forensics research [14], which had shown some promise for use in anomaly detection, but no discussion of feature analysis was mentioned. No other literature presents itself which persists tool outputs to a central data repository in order to farm for features to use in classification. Features were selected based on a suspicion that they may be good indicators for identifying malicious activity. Table 3.7 lists features evaluated for effectiveness in distinguishing between normal and attack events.



Name	Comment	Log File Type	Log File Name
Threads		Sequential...	C:\PerfLogs\trace_...

Current log file name:
C:\PerfLogs\trace_02281336.etl

This log begins when it is started manually.

[● Events logged by system provider:]

- [✓] Process creations/deletions
- [✓] Thread creations/deletions
- [✓] Disk input/output
- [✓] Network TCP/IP
- [✓] Page faults
- [✓] File details

○ Nonsystem providers:

Run As: <Default>

Figure 3.3: Use Performance Monitor to configure the event trace capture. For captures, the default system provider is used with all capture options selected.

Relevant fields captured from the event trace sensor include, but are not limited to those listed in Table 3.8.

Table 3.8: Event Trace for Windows features which are evaluated for effectiveness in distinguishing between normal and attack events.

Event Trace for Windows Features evaluated	
Field Name	**Description**
# Operations	Count of events
# Event IAT	Avg Event Inter Arrival Time
Tot User Time(ms)	Sum User time (in milliseconds)
Tot Kernel Time(ms)	Sum Kernel time (in milliseconds)
# threads at work	Count Unique threads performing operations
# Bytes In	Sum Bytes Received (TCP/UDP Events)
# Bytes Out	Sum Bytes Sent (TCP/UDP Events)
# Disk IO Read	Count of events
# Disk IO Write	Count of events
# File IO Create	Count of events
# File IO Name	Count of events
# Page Fault CopyOnWrite	Count of events
# Page Fault DemandZeroFault	Count of events
# Page Fault HardPageFault	Count of events
# Page Fault TransitionFault	Count of events
# Process End	Count of events
# Process Start	Count of events
# TcpIp Disconnect	Count of events
# TcpIp Reconnect	Count of events
# TcpIp Retransmit	Count of events

Continued on Next Page...

Table 3.8 – Continued	
Field Name	**Description**
# TcpIp Send	Count of events
# TcpIp Recv	Count of events
# TcpIp end	Count of events
# Thread End	Count of events
# Thread Start	Count of events
# Thread End	Count of events
# Thread End	Count of events
# Thread End	Count of events
# UdpIp Send	Count of events
# UdpIp Recv	Count of events
# Other Event	Count of events not listed above
Abnormal Ports accessed	Based on “Normal” data capture
Abnormal Files accessed	Based on “Normal” data capture

3.1.3 Sensor View of System. In Figure 3.4, an abstract view of the forensic segments of a Windows XP Workstation, and the coverage of these segments by the sensors selected is shown. First, the diagram reflects that TShark.exe monitors network activity. Then, pslist.exe and listdlls.exe provide meta data about running processes and their associated dlls. Next, ETW provides events pertaining to file and registry accesses (among other things). Additionally, PSloglist provides the capability to export system, security and application logs. LogonSessions.exe provides information pertaining to active sessions and the processes associated to them. Finally, TCPVCon provides a way to relate port activity with process information gathered. This coverage is desired in order to cast a wide net around potential features which may indicate malicious activity.

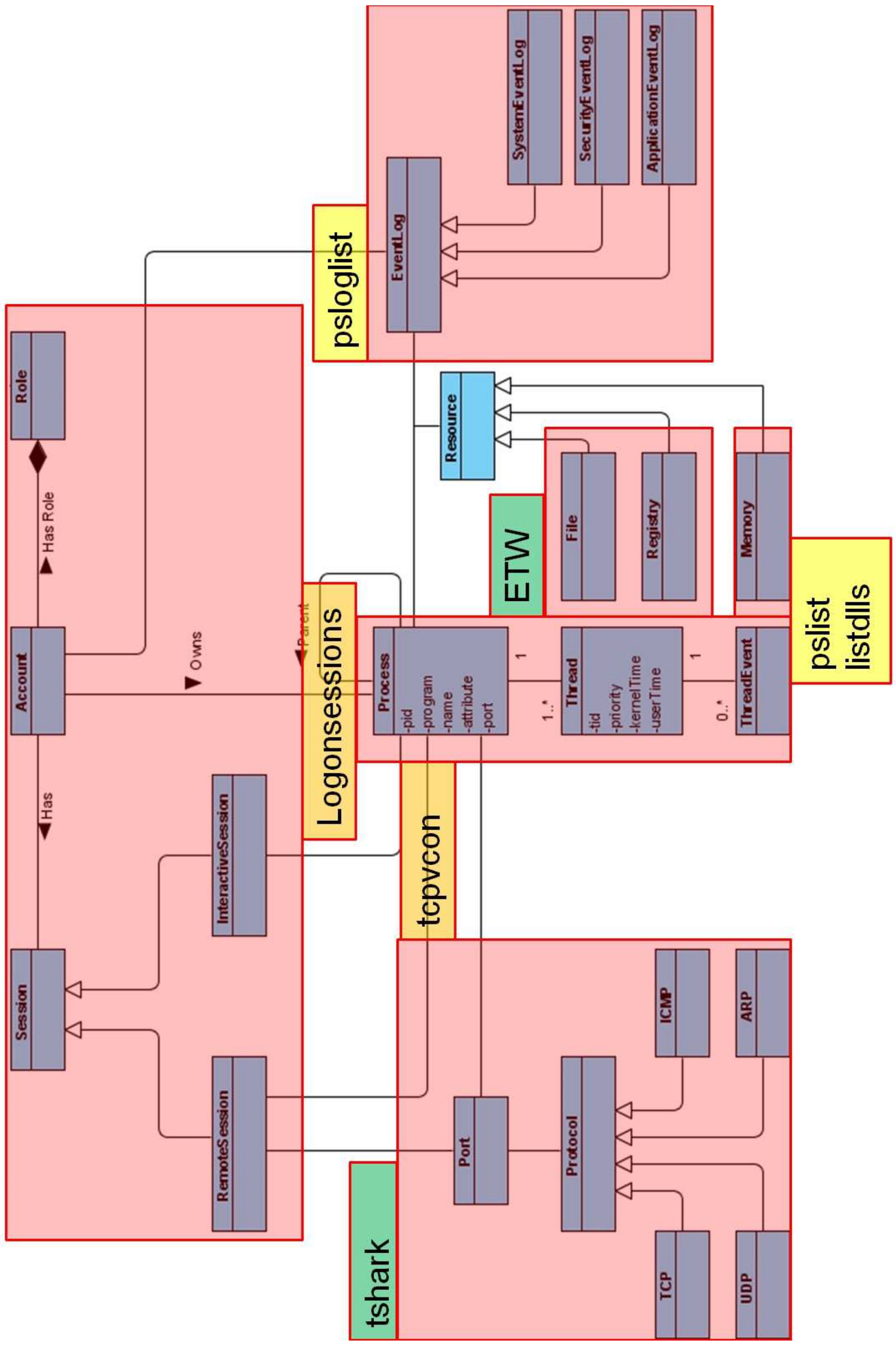


Figure 3.4: Forensic segments of a Windows XP workstation, and the sensors which can monitor them.

3.2 CRISP-DM: Preparation

To describe and explain the data, a framework for parsing the sensor data in a consistent and repeatable way is developed in Java. Since each sensor's output is specific, a distinct parser exists for each sensor. Parsing data from the selected snapshot sensors is relatively straightforward, as the outputs are already in human-readable form (text). Each of these sensors produces structured output which can be parsed in a consistent manner. However, each of the selected transactional parsers generate binary files (ETL and PCAP files), which need to be parsed with the assistance of some other means before they can be read. Tracerpt.exe is used to translate ETL data from its raw form to text before attempting to parse and analyze the data. To parse the network packet data captured by tshark.exe, an open source Java SDK called jNetPcap [3] is used. jNetPcap is a Libpcap/WinPcap wrapper, so it is capable of parsing data produced by any packet capturing application which is built upon Libpcap/WinPcap, such as WireShark and tshark. The jNetPcap API is included in the parsing framework specifically for this purpose.

As mentioned previously, sensors are classified as either transactional or as snapshot sensors. A transactional sensor class was developed for each of the transactional sensors, which include summarizing methods in addition to the above listed methods.

3.2.0.1 Snapshot Parsers. Each transactional parser implements the I_SnapshotParser interface in Java shown below; each class implementing this interface must provide the capabilities listed here. Parameterized templates are used, denoted by T in the source code listing below; an implementing class must provide these methods, plugging in its own class name in place of the T operator. This method provides for extensibility for plugging in additional sensors to the framework later on.

```
package model.parser;
import java.io.File;
import java.util.ArrayList;
import model.ForensicDate;
```

```
public abstract interface I_SnapshotParser<T>
{
public abstract ArrayList<T> parse(File f);
public abstract ArrayList<T> getObsByFTW(ArrayList<T> arr, ForensicDate ftw);
public abstract void exportToCSV(ArrayList<T> arr, String s);
public abstract ArrayList<T> fillGaps(ArrayList<T> arr);
public abstract String getCSVHeaders();
public abstract String toCSV();
}
```

3.2.0.2 Transactional Parsers. Each transactional parser implements the I_TransactionalParser interface in Java shown below; each class implementing this interface must provide the capabilities listed here. Parameterized templates are used, denoted by T in the source code listing below; this delegates the class to implement a class-specific function, plugging in their own class name in place of the T operator.

```
package model.parser;
import java.io.File;
import java.util.ArrayList;
import model.ForensicDate;
public abstract interface I_TransactionalParser<T>
{
public abstract ArrayList<T> parse(File f);
public abstract ArrayList<T> filterByFTW(ArrayList<T> arr, ForensicDate ftw);
public abstract void exportToCSV(ArrayList<T> arr, String s);
public abstract String getCSVHeaders();
public abstract String toCSV();
public abstract T summarize(ArrayList<T> arr);
public abstract ArrayList<T> buildExemplars(ArrayList<T> arr);
}
```

3.2.1 Data Alignment. In order to build a forensically diverse observation which includes data from all the sensors, a search of related features is required. Recall from Figure 3.4, the forensic coverage of the selected sensors has a little overlap; overlapping features are used to relate sensors from two or more sensors together as a single combined observation. Three elements are used to align data between sensors: Process ID, Local Port and Forensic Time Window.

Process ID. Each selected sensor, with the exception of the network packet capturing sensor, includes process id (PID) in their outputs. Unfortunately, four of these sensors capture snapshots only, and are unable to see processes which start and end between snapshots. Luckily, ETW captures process start and stop activity, as well as which executable file the process id is associated with; this information is used to partially fill forensic data gaps between snapshots. Because the Process ID is assigned at run time by the operating system, and is not guaranteed to map to an executable between captures, a standard process id (SPID) field is created. The SPID is an integer assigned to a specific executable which is observed in at least one of the captures, and becomes the key by which data is normalized later in the process.

Local Port. Tying network data provided by tshark to the process data is a bit of a challenge. Although tcpvcon.exe provides PIDs and their associated ports, it is a snapshot sensor, and subject to the same problem as other snapshot sensors. Fortunately, ETW once again provides a means to fill in some of the forensic data gaps between snapshots. FTW captures summary information about UDP and TCP events (by PID); this information is used to identify traffic which occurs between snapshots.

Forensic Time Window. Another data alignment consideration is timing. Because snapshot sensors are executed one after another, there are discrepancies in the collection times for the "current" snapshot of a process. This is addressed using forensic time windows (FTW), two-second time frames by which a process' or port's activity can be summarized. Shilland [62] takes this approach to account for variations in code speed on a MANET (MANET) IDS. Since collections are performed on one machine using a single computer's clock to track time, it is assumed time synchronization issues which arise from distributed systems [45] are not an issue.

3.2.2 Feature Selection. In order to perform some of the analysis, the parsed data is stored in a Microsoft SQL Server 2008 Server, Express edition. Data is uploaded en mass for labeling, normalization and for performing feature analysis.

Data is initially affixed with two labels per observation en mass according to its temporal proximity to a known event within a scenario. The first label is a category label, which is either "normal", "scanning" or "attack." The second label is an attack stage label, which is either "normal" or the generic name for an attack stage such as "planting back door" or "privilege escalation". Although it is known what activity takes place at a particular time t, it is unknown what low level impacts the activity has on other processes, such as causal relationships and interprocess communication. Therefore, all evidence collected at the time stamp is labeled similarly.

Once collected and labeled, the data understanding portion of the research begins. Again, the goal of refining data understanding is to selectively determine features from data collections which may help to distinguish malicious activity from normal activity. Because malicious activity comes in many forms, an attempt to correlate each feature to specific types of malicious activity: scanning, enumerating, gaining access is performed. And, because the intent is to pass these features over to a neural network for classification, a feature does not need to singularly and deterministically identify the class. However, a relevant feature is one which should generally provide some separation between classes.

With this in mind, features are identified using one of two methods: "Set distinction" and "Abnormal measures."

3.2.2.1 Set Distinction. The first method for identifying features is through set distinction. Datasets of distinct values occurring within a candidate feature are built using Transact SQL (T-SQL); one set is generated for each class of data to classify. From this, more T-SQL statements are executed to identify which values only occur in one collection or another. Results from this process are then manually examined to determine if the presence of a particular value is a matter of coincidence (actually part of normal operations) or appears correlated to a specific event or set of events. Items which appear correlated to a specific event are researched in literature to determine if the value has significant meaning to the event. From this, a decision is made to include the discovery of this value as a feature for HIDS or not.

3.2.2.2 Abnormal Measures. For features which rely on measures (e.g., counts of event x or Δ measures), an analysis of the distribution of the datapoints is performed. To do this, sample statistics are taken for each observed variable, namely its mean $\bar{x}$ as shown in Equation 3.1, sample standard deviation s as shown in Equation 3.2, skewness *skew* as shown in Equation 3.3 and kurtosis *kurt* as shown in Equation 3.4. These measures will help us to determine each variable's potential value toward separating classes of data.

The sample mean, $\bar{x}$, provides an average of all values within the sample, and is given as:

$$\bar{x} = \sum_{i=1}^{n} \frac{x_i}{n} \tag{3.1}$$

Standard deviation, s, is a measure of the average difference between each data point and $\bar{x}$, and is given as:

$$s = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}} \tag{3.2}$$

Skewness describes the level of asymmetry of the distribution of a random variable with regard to its mean. Positive skewness indicates a distribution with an asymmetric tail extending toward more positive values. Negative skewness indicates a distribution with an asymmetric tail extending toward more negative values. The formula for the skewness consists of sum of the deviations from the mean divided by standard deviation on the third degree.

$$skew = n/(\frac{n-1}{n-2}) * \sum_{i=1}^{n}(\frac{x_i - \bar{x}}{s})^3 \tag{3.3}$$

Kurtosis characterizes how flat or noisy a distribution is when compared to the normal distribution. Positive kurtosis indicates a peaked distribution, while negative kurtosis indicates flatter distribution. Kurtosis is the sum of the deviations from the mean divided by standard deviation to the fourth power.

$$kurt = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^4}{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2} - 3 \tag{3.4}$$

Upon completion of the attack sequence, forensic data is aggregated from each sensor on the Sensor client into two-second time intervals, grouped by key fields such as process id or local port number. Each time interval is assigned an ID which corresponds with the start date and time of the interval. Each record in the time interval is grouped by its key fields, and the remaining fields of interest are summarized via aggregation functions (averages, summations, deltas, mean, standard deviation and kurtosis) as they relate to the key fields.

3.2.3 Signature-Based Feature Identification. The first method used to generate features is performed by identifying the presence or absence of suspicious signatures within the forensic data. This method is implemented on variables which have no numerical measurement, e.g. string-based variables whose literal value has meaning in its own right.

From each class of labeled data to identify (e.g., 'normal', 'scanning', 'exploit'), a set of unique observations is built. Items which only occur within a single class, or occur relatively infrequently in other classes, become candidate class signatures. For each candidate class signature, a mechanism next interprets the relevance of the item's meaning with regard to the temporality of the observation within the dataset in question. The researcher evaluates each candidate signature independently to identify what data and/or capabilities the file provides, and the inferred meaning of accessing the file at time t. Additionally, the human operator compares instances of the signature detected against collections to determine if the item is a coincidental instance or correlated instance to the activity at the given point in time. Finally, if an item is deemed relevant to the class of data to identify, it is added as a feature to look for within future collections. Validation of these features includes attempts to produce the signature in a data collection through the performance of valid actions.

3.3 CRISP-DM: Modeling and Analysis

An artificial neural network (ANN) is used to model the data set of selected features. Each observation is formatted appropriately for use with the MATLAB Neural Network Toolbox, to include the observation's true classification label and a vector of the selected features. The input data is split into two groups, populated via monte carlo sampling [58]: two-thirds for training and one-third for testing. The ANN is trained via back propagation using varying training parameters:

- Learn Rate: vary from 0.01 to 0.2
- Activation Function: vary between tansig, logsig [29]
- Number of Hidden Layers: vary from 1 to 3
- Number of Neurons in Hidden Layers: vary from 3 to 30
- Target Accuracy Rate (MSE): 0.0001
- Maximum Epochs: 1,000

Once the ANN training phase is completed, the ANN is used to classify the testing data set. A ROC Curve Plot and a confusion matrix are both used to interpret the classification accuracy of the ANN model.

3.4 Summary

A research methodology is now laid out by which to study the feasibility of behavior-based classifiers for network attacks. An understanding of research goals and hypothesis, a description of the development and test environments, and a description of the experiments along with assumptions and limitations, is critical for anyone intending to duplicate or continue this research. The results of the experiments can be found in Chapter four.

IV. Results & Analysis

The data collections were performed in three phases, and produced three datasets: Normal, Scanning and Exploit. Table 4.1 provides a summary of these collections in terms of execution time, number of files generated and cumulative raw data size.

4.1 Signatures Discovered

Collection analysis identified seven specific items that correlated well with the events occurring at the time and provided discrimination between normal and attack. Each feature was discovered independently by performing data discovery activities on the data, querying data for correlation between events and observations within the data. Six of the seven features are attributed to File IO activity, the seventh is a measure of port activity. Admittedly, a few of these features may not perform well in environments which differ wildly from the environment used in this research.

1. *High count of local ports with activity.* In general, 20 or more distinct active ports indicated the presence of port scanning activity within the data collected. This feature is a volumetric measure against normal activity, which peaked at 25 active ports while in normal stage [17], web surfing. The threshold can be set higher, but at the risk of producing more false negatives within the data. Additionally, it is completely feasible to be scanned via low and slow scanners, which would not report above the threshold established. This is a problem which plagues many anomaly detection systems. Lower level protocol analysis would be required to more definitively identify scanning activity as a signature,

Table 4.1: Summary of Collection times, number of files and the amount of raw data generated. Collection times are intended to simulate a long period of probing followed by a direct and quick set of exploits which take advantage of learned vulnerabilities.

Collection	**Minutes**	**Files**	**Raw Data Size**
Normal	41 mins	2803	1.18 GB
Scanning	40 mins	2561	904 MB
Exploit	9 mins	244	131 MB

but signatures of activity are not without error. As shown in the collection, a high incident of false positives resulted from protocol analysis using signatures.

2. *File IO activity involving a /mailslot/.* Mailslots [60] support unreliable unidirectional data transmission, and support the capability to receive broadcasts; however, a valid application for /mailslot/ activity is time synchronization. The collections performed did not include a time server, so results may be biased for this reason. Only four events across all collections yielded /mailslot/ activity, two events had four counts of this activity and four events were attributed to times when the attacker was using a remote connection to execute commands on the target machine. Two events had one count of this activity, which is tied to a specific tool, covered next.

3. *File IO activity involving /mailslot/Nessus.* Two events related to Nessus vulnerability scanning involved this activity. As mentioned previously, mailslots provide a way for receiving broadcasts. Nessus may use mailslot broadcasts when looking for vulnerabilities across a number of machines on the same network or subnet.

4. *File IO activity involving /mup/.* Multiple Universal Naming Convention (UNC) Providers (MUP) [60] are device drivers which field input/output requests to files or devices that are exposed to the network via UNC. 23 instances of MUP signatures occur across seven events from the attack collection, and nowhere else in any other collection. Valid MUP usage include accessing a shared directory on the target system. However, since the test operates under the assumption that UNC shares are disallowed by policy on a workstation, MUP should not be seen through normal usage.

5. *File IO activity involving net.exe* Net.exe and net1.exe each provide networking services via the command line. However, more than two-thirds of all net.exe and net1.exe calls made happened during two events in the capture, one where metasploit's adduser payload was launched at the target, the other when an

attack script created a network drive connection back to the attackers machine. This feature obviously cannot be used by itself, as there are a number of valid user situations where net.exe may be used; an alert based solely on this would, over time, result in a high incidence of false positives.

6. *Authentication File Dump Feature* This is a composite condition, where one of the following was noted in File IO activity: regback, lsadump, cachedump, fgdump-log. Presence of each of these strings was related to at least one of the attempts to exfiltrate passwords, but is tied to specific tools. System administrators may have reasons for wanting to run these utilities, but the typical user should not be running these applications.

7. *Remote as System feature* This string is the most interesting of them all, it involves an embedded string encoding of the System account's session id (which is hard-coded into the operating system as decimal 999, or 0x3e7). A typical string that would trigger the remote as system feature is:
"\Device\LanmanRedirector\;P:00000000000003e7\blackhat\shareddocs\exploit\"
Originally, the feature selected was:
"\device\lanmanredirector."
However, this proved a poor performer, since a valid user with a session on the machine can trigger that rule by accessing any of their connected drives. In fact, the user's session id would be present where the system session id is in the sample string above. The correlation between this string and events at the time indicate that its presence signifies remote access to the machine with System credentials.

4.2 Neural Network Results

The seven features identified within the data were used to train a neural network using back propagation. Data was split into two groups, populated via monte carlo sampling [58]: two-thirds for training and one-third for testing. The training data was used to train the neural network with learn rates ranging from 0.1% to 2%,

Table 4.2: Upon simulating the trained network against unbiased test data, results show the the features selected perform very well to identify normal and malicious activity.

Output Class	Normal	Scanning	Exploit	
Normal	885 (95.1%)	2 (0.2%)	0 (0.0%)	99.8% (**0.2%**)
Scanning	0 (0.0%)	32 (3.4%)	0 (0.0%)	100% (**0.0%**)
Exploit	0 (0.0%)	0 (0.0%)	12 (1.3%)	100% (**0.0%**)
	100% (**0.0%**)	94.1% (**5.9%**)	100.0% (**0.0%**)	99.8% (**0.2%**)

Target Class

and with various activation functions (e.g., tangent sigmoid or log sigmoid activation functions [29]), a limit of training 1000 epochs, and a MSE goal of 10^{-4}. Seven input neurons made up the input layer, corresponding to the seven features. Three output neurons made up the output layer, corresponding to the number of classes to identify. Various hidden layer attributes were used, varying between 1-3 hidden layers, with 5-20 neurons per layer. Multiple iterations were accomplished using new training and testing groups. Results from performance analysis consistently settled to 10^{-2} or better. The best parameters for training this neural network appear to be using two hidden layers, with 3, then 7 neurons, respectively. The performance plot in Figure 4.2, ROC curve plot in Figure 4.2, and confusion matrices in Table 4.2illustrate these results.

The selection of port counts with activity was not an effective feature with regard to a neural network. If a rule is established to signal an alert when a machine has 25 or more connections, the datasets collected would have yielded only 3 false alerts, but would have identified 100% of the Nessus and NMAP scanning events.

Second, the identified signatures which indicate malicious activity (other than scanning) may also be either present or not present; errors arise when feeding these to

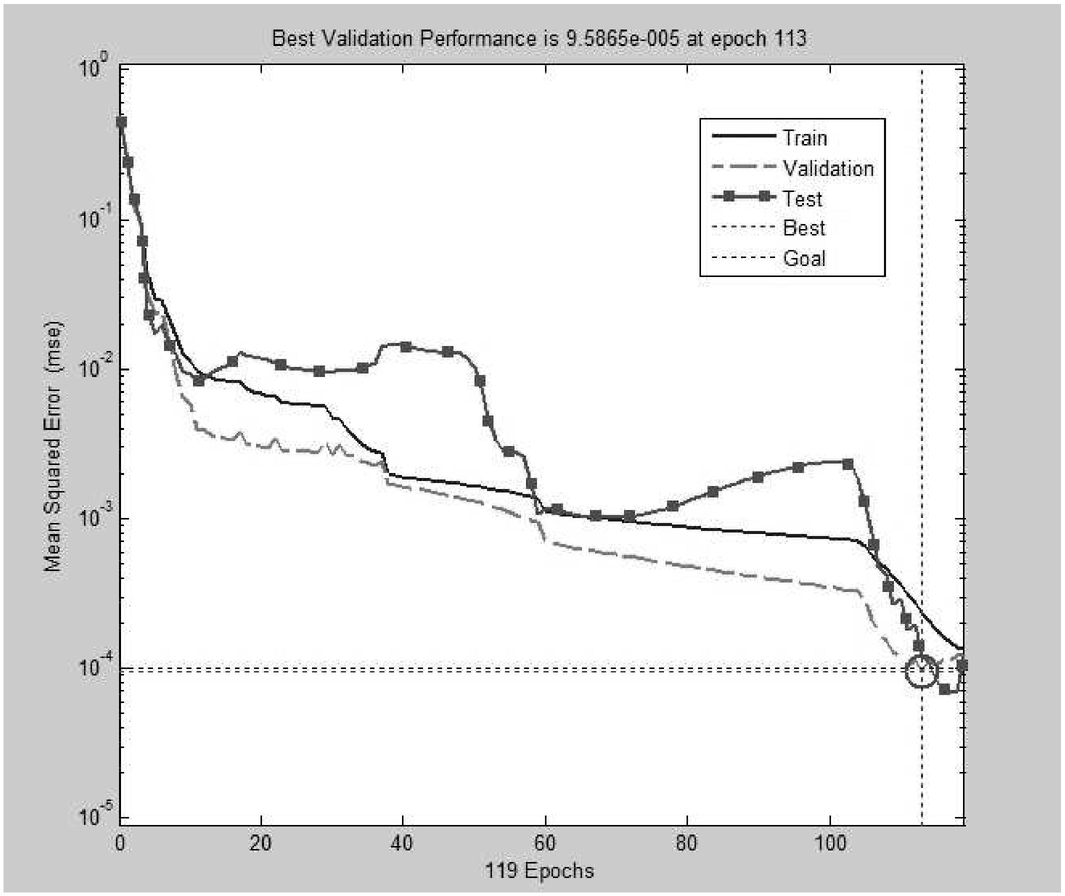


Figure 4.1: The mean squared error of the trained neural network-based classifier consistently converges to a MSE of $10^{-3.5}$ or better, which indicates the features selected perform very well overall for classifying normal and malicious host-based activity within the collections gathered.

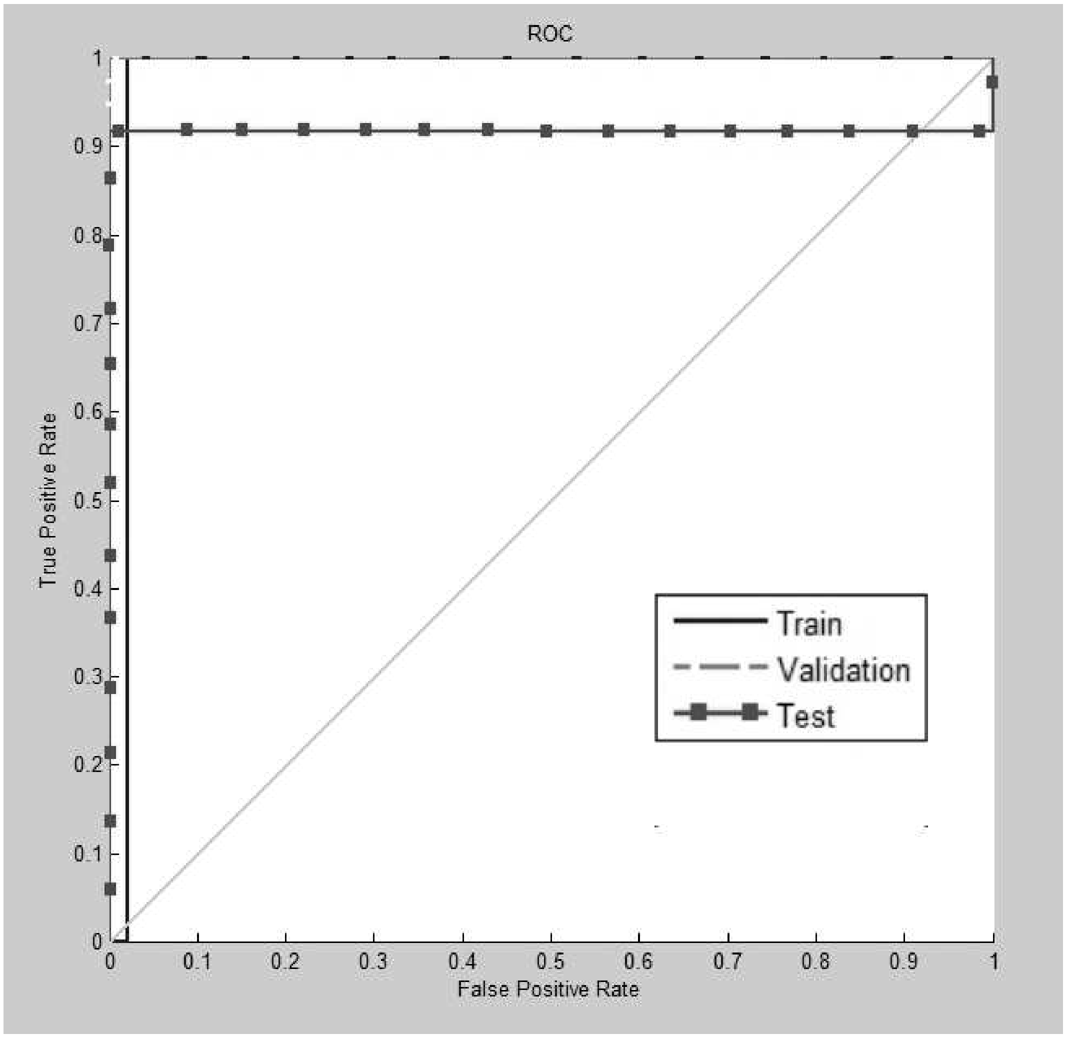


Figure 4.2: Upon simulating the trained network against unbiased test data, results show the features selected perform very well to identify normal and malicious activity.

a neural network since the incident rate may be too low for the network to learn. A larger collection would be necessary, along with more training in order to learn these features. However, if the features identified from labeling data as malicious when there may be a low normalized number in the data.

4.2.1 Alerts Triggered: Data Understanding ANN. The seven rules learned through data understanding led to 226 true positive alerts triggered through the ANN. Using the rules learned through data understanding, there were 10 false negative alerts and 16 false positive alerts from across the three collections. Tables 4.3 - 4.5 provide summarized views of the alerts triggered by the data understanding ANN, along with the alert's relation to events during the controlled collections.

Table 4.3: The data understanding ANN generated 3 alerts, 3 false positive alerts, and 0 false negatives, pertaining to the target machine's network activity during 40 minutes of what is considered normal activity. While the events were related to valid user activity, the threshold for the number of active ports was likely set too low, resulting in the false identification of scanning activity.

Data Understanding ANN Alerts - Normal Activity Collection			
Alerts	**Alert Trigger**	**Classification**	**EVENT**
3	20+ active local ports	Scanning	[17] Surfing Web, 43 ports
1 Total alerts in "Normal Activity" collection			

Table 4.4: The data understanding ANN generated 85 alerts, with 4 false positives and 11 false negatives (10 of which were false negatives for Snort as well), pertaining to the target machine during the 40 minutes of scanning activity intermixed with periods of normal activity.

Data Understanding ANN Alerts - Scanning Activity Collection			
Alerts	**Alert Trigger**	**Classification**	**EVENT**
1	File IO with net.exe	Exploit	[n/a] Between scanning events
3	20+ active local ports	Scanning	[n/a] Surfing Web
1	20+ active local ports	Scanning	[10] NMap, Intense Scan
2	20+ active local ports	Scanning	[11] NMap, Intense Scan plus UDP
28	20+ active local ports	Scanning	[12] NMap, Intense Scan plus TCP
2	20+ active local ports	Scanning	[13] NMap, Intense Scan, No Ping
1	20+ active local ports	Scanning	[15] NMap, Quick Scan
1	20+ active local ports	Scanning	[16] NMap, Quick Scan Plus
2	20+ active local ports	Scanning	[18] NMap, Regular Scan
2	20+ active local ports	Scanning	[19] NMap, Slow Comprehensive Scan

Continued on Next Page...

Table 4.4 – Continued			
Alerts	**Alert Trigger**	**Classification**	**EVENT**
40	20+ active local ports	Scanning	[20] Nessus, Aggressive Scan
2	20+ active local ports	Scanning	[21] Nessus, Stealthy Scan
85 Total alerts in "Scanning Activity" collection			

Table 4.5: The data understanding ANN generated 164 alerts, with 3 false positives and 5 false negatives (3 of which were false negatives for Snort as well) pertaining to the target machine during the nine minute collection period of intermixed normal and exploit activity.

Data Understanding ANN Alerts - Exploit Activity Collection			
Alerts	**Alert Trigger**	**Classification**	**EVENT**
3	File IO with net.exe	Exploit	[n/a] No event at this time
8	File IO with net.exe	Exploit	[1] metasploit: adduser payload
17	File IO with net.exe, MUP, mailslot or remote as system	Exploit	[3] RShell: connect network drive from target to hacker machine.
2	File IO with Remote as system	Exploit	[4] RShell: interacting with console.

Continued on Next Page...

Table 4.5 – Continued

Alerts	Alert Trigger	Classification	EVENT
15	File IO with net.exe, MUP or remote as system	Exploit	[5] RShell: run addAdminAccount.bat
50	File IO with MUP, password dump or remote as system	Exploit	[6] RShell: run stealpasswords.bat
9	File IO with MUP, Remote as system	Exploit	[7] RShell: run plantBackdoor.bat
4	File IO Remote as system	Exploit	[11] RShell: run plantBackdoor.bat
15	File IO MUP, mailsot, remote as system	Exploit	[13] RShell: run enumusersingroups.vbs
7	File IO MUP and Remote as system	Exploit	[14] RShell: run covertracks.bat
18	File IO MUP, remote as system	Exploit	[15] RShell: run plantkeylogger.bat
18	File IO MUP, remote as system	Exploit	[15] RShell: run covertracks.bat

4.2.2 Alerts Triggered: Snort IDS. Results were compared to the leading open source intrusion detection system provided by SourceFire, Snort [64]. Snort was installed on a Linux virtual machine (Ubuntu version 8.10), configured with the default rules. Tables 4.6–4.8 provide summarized views of the alerts triggered by Snort for each collection, along with the alert's relation to events during the controlled collections.

Table 4.6: Snort generated 285 alerts, with 285 false positives and 0 false negatives, pertaining to the target machine during 40 minutes of what is considered normal activity for just one machine.

Snort Alerts - Normal Activity Collection			
Alerts	**Alert Trigger**	**Classification**	**EVENT**
6	COMMUNITY WEB-MISC mod_jrun overflow attempt	Web Application Attack	[16] Powerpoint insert online clipart [17] Internet Explorer use
235	MISC UPnP malformed advertisement	Misc Attack	[n/a] Valid alerts, but not related a scripted event
12	SCAN UPnP service discover attempt	Generic Protocol Command Decode	[n/a] Valid alerts, but not related a scripted event
11 2	WEB-MISC Lotus Notes .exe script source download attempt WEB-MISC Invalid HTTP Version String	Web Application Attack Detection of a non-standard protocol or event	[17] Internet Explorer use [17] Internet Explorer use

Continued on Next Page...

Table 4.6 – Continued

Alerts	**Alert Trigger**	**Classification**	**EVENT**
6	NETBIOS SMB Session Setup NTMLSSP unicode asn1 overflow attempt	Generic Protocol Command Decode	[2] Outlook short email [9] Word [n/a] One event between steps
1	ICMP Destination Unreachable Communication Administratively Prohibited	Misc activity	[17] Internet Explorer use
10	NETBIOS SMB IPC$ unicode share access	Generic Protocol Command Decode	[2] Outlook short email [9] Word [17] Internet Explorer use [20] Word, Excel, Powerpoint together [n/a] One event between steps
1	(http_inspect) U ENCODING	(not listed)	[17] Internet Explorer use
1	(portscan) UDP Portsweep	(not listed)	[17] Internet Explorer use
285 TOTAL ALERTS IN "NORMAL ACTIVITY"			

Table 4.7: Snort generated 561 alerts, with 4 false positives and 9 false negatives, pertaining to the target machine during 40 minutes of what intermixed scanning and normal activity for just one machine.

Snort Alerts - Scanning Activity Collection			
Alerts	**Alert Trigger**	**Classification**	**EVENT**
4	COMMUNITY WEB-MISC mod_jrun overflow attempt	Web Application Attack	[n/a] Valid alerts, but not related a scripted event
6	SCAN nmap XMAS	Attempted Information Leak	[10] NMAP intense scan [11] NMAP intense scan + UDP [12] NMAP intense scan + TCP [13] NMAP intense scan no ping [16] NMAP quick scan plus [19] NMAP slow progressive scan
6	(snort_decoder): Tcp Window Scale Option found with length ¿ 14		[10] NMAP intense scan [11] NMAP intense scan + UDP [12] NMAP intense scan + TCP [13] NMAP intense scan no ping [16] NMAP quick scan plus [19] NMAP slow progressive scan
Continued on Next Page...			

Table 4.7 – Continued

Alerts	Alert Trigger	Classification	EVENT
205	MISC UPnP malformed advertisement	Misc Attack	[10] NMAP intense scan (11 alerts) [12] NMAP intense scan + TCP [15] NMAP quick scan [20] NESSUS aggressive scan [21] NESSUS lighter scan
6	SNMP public access udp	Attempted Information Leak	[20] NESSUS aggressive scan [21] NESSUS lighter scan
2	SNMP private access udp	Attempted Information Leak	[20] NESSUS aggressive scan [21] NESSUS lighter scan
12	SNMP request udp	Attempted Information Leak	[11] NMAP intense scan + UDP [19] NMAP slow progressive scan [20] NESSUS aggressive scan [21] NESSUS lighter scan

Continued on Next Page...

Table 4.7 – Continued

Alerts	Alert Trigger	Classification	EVENT
6	SNMP request tcp	Attempted Information Leak	[11] NMAP intense scan + UDP [12] NMAP intense scan + TCP [13] NMAP intense scan no ping [18] NMAP regular scan [20] NESSUS aggressive scan [21] NESSUS lighter scan
2	SNMP trap udp	Attempted Information Leak	[11] NMAP intense scan + UDP [19] NMAP slow progressive scan
4	SNMP trap tcp	Attempted Information Leak	[12] NMAP intense scan + TCP [20] NESSUS aggressive scan [21] NESSUS lighter scan
7	SNMP AgentX/tcp request	Attempted Information Leak	NMAP and Nessus (steps 10, 12, 13, 18, 20, 21)
12	TFTP Get	Potentially Bad Traffic	[20] NESSUS aggressive scan [21] NESSUS lighter scan

Continued on Next Page...

Table 4.7 – Continued			
Alerts	**Alert Trigger**	**Classification**	**EVENT**
2	MISC AFS access	Misc activity	[20] NESSUS aggressive scan [21] NESSUS lighter scan
2	MISC xdmcp info query	Attempted Information Leak	[20] NESSUS aggressive scan [21] NESSUS lighter scan
17	SCAN UPnP service discover attempt	Detection of a Network Scan	[20] NESSUS aggressive scan [21] NESSUS lighter scan
2	MS-SQL ping attempt	Misc activity	[20] NESSUS aggressive scan [21] NESSUS lighter scan
2	NETBIOS DCERPC Remote Activation bind attempt	Attempted Administrator Privilege Gain	[20] NESSUS aggressive scan [21] NESSUS lighter scan
101	NETBIOS SMB-DS Session Setup AndX request unicode username overflow attempt	Attempted Administrator Privilege Gain	[20] NESSUS aggressive scan [21] NESSUS lighter scan
79	NETBIOS SMB-DS IPC$ unicode share access	Generic Protocol Command Decode	[20] NESSUS aggressive scan [21] NESSUS lighter scan

Continued on Next Page…

Table 4.7 – Continued

Alerts	**Alert Trigger**	**Classification**	**EVENT**
2	NETBIOS SMB-DS D$ unicode share access	Generic Protocol Command Decode	[21] NESSUS lighter scan
2	NETBIOS SMB-DS C$ unicode share access	Generic Protocol Command Decode	[21] NESSUS lighter scan
4	NETBIOS SMB-DS ADMIN$ unicode share access	Generic Protocol Command Decode	[21] NESSUS lighter scan
3	DDOS mstream client to handler	Attempted Denial of Service	[12] NMAP intense scan + TCP [20] NESSUS aggressive scan [21] NESSUS lighter scan
9	NETBIOS SMB-DS repeated logon failure	Unsuccessful User Privilege Gain	[20] NESSUS aggressive scan
6	NETBIOS SMB Session Setup NTMLSSP unicode asn1 overflow attempt	Generic Protocol Command Decode	[1] ws_ping [10] NMAP intense scan [n/a] 4 alerts between events
10	NETBIOS SMB IPC$ unicode share access	Generic Protocol Command Decode	[1] ws_ping [10] NMAP intense scan [n/a] 4 alerts between events

Continued on Next Page...

Table 4.7 – Continued

Alerts	Alert Trigger	Classification	EVENT
2	NETBIOS SMB-DS Session Setup NTMLSSP unicode asn1 overflow attempt	Generic Protocol Command Decode	[20] NESSUS aggressive scan [21] NESSUS lighter scan
2	NETBIOS DCERPC IActivation little endian bind attempt	Generic Protocol Command Decode	[20] NESSUS aggressive scan [21] NESSUS lighter scan
2	ICMP L3retriever Ping	Attempted Information Leak	ws_ping and NMAP (steps 1, 10)
6	ICMP PING NMAP	Attempted Information Leak	[20] NESSUS aggressive scan [21] NESSUS lighter scan
1	ICMP traceroute ipopts	Attempted Information Leak	[20] NESSUS aggressive scan
4	BAD-TRAFFIC tcp port 0 traffic	Misc activity	[20] NESSUS aggressive scan [21] NESSUS lighter scan
4	SCAN Amanda client version request	Attempted Information Leak	[20] NESSUS aggressive scan [21] NESSUS lighter scan
1	WEB-CGI wrap access	Attempted Information Leak	Between scan events
2	(http_inspect) OVERSIZE REQUEST-URI DIRECTORY		Between scan events

Continued on Next Page...

Table 4.7 – Continued			
Alerts	**Alert Trigger**	**Classification**	**EVENT**
1	(http_inspect) DOUBLE DECODING ATTACK		Between scan events
15	(portscan) TCP Portscan		[1] ws_ping [10] NMAP intense scan [12] NMAP intense scan + TCP [13] NMAP intense scan no ping [15] NMAP quick scan [17] NMAP quick tracert [18] NMAP regular scan [20] NESSUS aggressive scan [21] NESSUS lighter scan
5	(portscan) UDP Portscan		NMAP and Nessus (steps 11, 19, 20, 21)
3	(portscan) UDP Portsweep		NMAP one one Nessus (steps 10, 19, 20)
561 TOTAL ALERTS IN ”SCANNING ACTIVITY”			

Table 4.8: Snort generated 57 alerts, 0 false positive alerts, and 6 false negative alerts pertaining to the exploit collection.

Snort Alerts - Exploit Activity Collection			
Alerts	**Alert Trigger**	**Classification**	**EVENT**
# Alerts	**Alert Trigger**	**Classification**	**EVENT**
42	MISC UPnP malformed advertisement	Misc Attack	[n/a] Valid alerts, but not related a scripted event
2	NETBIOS SMB Session Setup AndX request unicode username overflow attempt	Attempted Administrator Privilege Gain	[2] Metasploit reverse shell start, [3] Metasploit net use p: back to blackhat box
2	NETBIOS SMB-DS IPC$ share access	Generic Protocol Command Decode	[1] Metasploit adduser payload, [2] Metasploit reverse shell start
2	ICMP L3retriever Ping	Attempted Information Leak	[2] Metasploit reverse shell start
4	NETBIOS SMB IPC$ unicode share access	Generic Protocol Command Decode	[5] Metasploit rshell addadminaccount.bat
1	ATTACK-RESPONSES Microsoft cmd.exe banner	Successful Administrator Privilege Gain	[9] Netcat session started
2	ATTACK-RESPONSES directory listing	Potentially Bad Traffic	[n/a] Valid alerts, but not related a scripted event
Continued on Next Page...			

Table 4.8 – Continued			
Alerts	**Alert Trigger**	**Classification**	**EVENT**
2	NETBIOS SMB Session Setup NTMLSSP unicode asn1 overflow attempt	Generic Protocol Command Decode	[15] Netcat plantkeylogger.bat
57 TOTAL ALERTS IN ”ATTACK ACTIVITY”			

Table 4.9: Comparison between Snort and ANN Ruleset false positive and false negative alerts.

Dataset	**Ruleset**	**True Positive Alerts**	**False Positive Alerts**	**False Negative Alerts**
Normal	Snort	0	285	0
	ANN	0	3	0
Scanning	Snort	548	4	9
	ANN	70	4	11
Exploit	Snort	57	0	6
	ANN	156	3	5
Total	Snort	605	289	15
	ANN	226	10	16

Overall, the methodology presented in this thesis resulted just 10 false positives and 16 false negatives, a 91.5% reduction when compared to Snort IDS' 289 false positives and 15 false negatives. Additionally, six events involving malicious remote shell connections were detected through the features discovered via data understanding which were not detected by Snort, and three times as many true positive alerts for exploit activity were generated by the data understanding ANN.

With regard to scanning, it is noted that Snort reported significantly more true positive alerts than the ANN method. Conversely, the ANN method reported significantly more true positive exploit activity alerts. As a NIDS, Snort signals an alert for each packet meeting a specific signature, such as xmas tree scan or syn flood. The focus of this body of work was not to duplicate all the alerts that another system can create, but to discover a set of forensic data features which indicate a genre of events while minimizing false alerts. Also, recall that the data is summarized by two-second timeframe (see Section 3.2.1) with the ANN method for data alignment, which will naturally yield lower number of alerts. While the numbers appear skewed, a better comparison is between false positive and false negative alerts, as this is where one tool or the other misidentified normal activity as scanning or exploitation activity, or vice versa.

A system administrator who were to rely solely on Snort to perform intrusion detection would need to filter through 289 alerts to find the truly malicious traffic, a laborious and time consuming task for analyzing just 90 minutes of activity. Additionally, if this administrator were to miss the one alert which indicated the beginning of a reverse shell connection, no additional alerts would have tipped off the administrator of a potential intrusion. The data understanding ANN identified six exploit events which Snort did not alert at all, and a high number of alerts during those time periods.

Nine of the sixteen false negatives revolve around events which Snort had also not identified as malicious scanning events: lightweight scanning which falls below the thresholds developed for identifying scanning events, such as produced via ping or traceroute routines. The remaining false negatives resulted from the lack of correlated forensic evidence within the data collected. Of the ten false positives, six related to two periods of legitimate web browsing, where the maximum number of simultaneously active ports threshold was surpassed. The remaining four resulted from the presence of net.exe or net1.exe within a File IO event. Thresholds for a counts of each of these events could reduce false positive rates, but at the risk of producing additional false negatives. Additional data collections can help to establish better baselines for thresholds, but a small number of false positives or false negatives is unavoidable.

V. Conclusions

This body of work delivers a methodology for collecting, parsing and analyzing live forensic data for the purpose of identifying relevant host-based intrusion detection features. An instrumented environment was established to perform host-based forensic data collections during periods of normal, scanning and exploit activity. An extensible Java-based framework was developed to parse sensor data from six sensors into sets of features for analysis. A SQL server database was built to aggregate and summarize forensic records, as well as identify telling features of an intrusion. In order to test the effectiveness of the selected features, an ANN was trained to classify normal, scanning and exploit activity based on selected features. The instrumented environment, sensor-parsing framework, SQL server database and ANN are all re-usable components for future work in this or related areas.

This research identified seven host-based intrusion detection features, which not only aided in detection of malicious activity, but also greatly reduced the incidence of false positive alerts when compared to Snort IDS. These features are:

1. *High count of local ports with activity.*
2. *File IO activity involving a /mailslot/.*
3. *File IO activity involving /mailslot/Nessus.*
4. *File IO activity involving /mup/.*
5. *File IO activity involving net.exe*
6. *Authentication File Dump Feature*
7. *Remote as System feature*

Using only this small set of features, more accurate and sustained reporting of malicious events occurred, though both the identification of events not discovered by Snort IDS, and the reduction of false positive alerts which must be filtered by a network defense technician. The methodology presented in this thesis identified six events of sustained malicious remote activity which Snort IDS had not generated alerts for. Additionally, the methodology yielded just 26 false positive alerts; a 91% reduction to Snort's 298 false positive alerts generated from the same collections.

Data understanding can be applied to host based intrusion detection research to minimize false reporting of malicious events, easing the administrative burden to network defenders, and providing for better decision quality information as to the events taking place on the host system. Additionally, the results of a study performed via data understanding can aid in the development of more effective and efficient live forensic sensors. By choosing what data to attend to, such a study can identify the minimum set of features a sensor needs to detect a malicious event, resulting in a lighter impact to system resources by the sensor.

5.1 Need for Sustained feature discovery research

Communication systems, their vulnerabilties and the threats which challenge their confidentiality, integrity, and availability, are constantly evolving. Neglecting to search through forensic evidence to identify features which help to detect these threats will further divide human trust in computing systems. What if instead of focusing on the signature of a specific threat, developers created IDS which set their sights on common forensic tripwires an attacker would disturb in order to successfully attack a network; the behavior of programs vs what programs physically look like at the byte level. Years of threat modeling research have solidified the definition of key behavioral building blocks believed to be foundational for successful network attacks. Depending on the goal(s) of the attack vector, one or more of these building blocks are used to compromise a system: footprinting, scanning, enumerating, gaining access, escalating privilege level, pilfering, covering tracks, creating back doors or executing a denial of service. If detection is focused toward reasoning about which stage an attack is in, the right COA (Course of Action) to mitigate the threat. The earlier this detection is accomplished, the better chances system administrators have to intercept andmitigate the threat before it can cause irrevocable damage.

5.2 Future Works

As mentioned in Chapter 1, there are a number of assumptions and known limitations for this research. Among these were the fact that collections were done in a controlled environment, the tools chosen for collecting forensic data consumed a lot of system resources, the intractability of the search space, and optimality conditions.

This research was done in a relatively closed environment, with minimal outside interference in order to know what events are transpiring. This was important for the identification of features, as it enabled the research to focus on data surrounding malicious activity as compared to data which were assumed normal activity periods. However, future efforts to test the effectiveness of features identified data understanding could involve collections in larger, more operationally realistic network environments. Other attacks and attack tools could be used to further expand and validate the set of identified features deemed relevant to detecting malicious activity. These attacks could include the launching of denial of service, virus, botnet and rootkit attacks.

Another potential future work is the development of lightweight forensic sensors, such that live collection has minimal impact on the system being monitored. Such a sensor could monitor for the features discovered by this and related research efforts, and would likely involve the development of low level methods, such as intercepting system and Application Program Interface (API) calls and other inter-process communication. There are other areas which can be explored using data understanding which were not explored through this research effort, such as performing memory captures or monitoring other I/O ports or specific applications for malicious activity.

As mentioned, the intrusion detection search space is humongous. While an analysis was performed to find relevant single-element forensic features within the forensic evidence, it was not an exhaustive search. A more exhaustive search of permutations between any/all forensic data elements may yield better results and identify new features which can be used to improve detection rates and work to further minimize false negative alerts. Additional permutations could be developed number of different ways, to include combinations of features for a given observation, timing and flow-based analysis between multiple observations.

Appendix A. Forensic Toolkit

A.1 Tools

The sensors listed in Table A are used to capture live forensic data for later analysis.

A.2 Scripts

The following scripts are written as a batch to launch forensic tools in a consistent and repeatable manner. Although care was taken to choose tools which minimized their impact to system resources, there is a noticeable performance hit (memory and processor) for all the data captures to files. During captures, with no additional activity other than sensors running, processor utilization hovered around 35% and memory utilization was around 185MB. The search for lightweight live forensic tools is a challenge in its own right, and should be accomplished prior to any similar research efforts.

The first file in the toolkit is the main "controller" script, named Autoexec.vbs. The controller is written in *Microsoft Visual Basic Script*, and is run from the command line and interpreted with *Windows Script Host v5.6* (included with Microsoft Windows XP, SP2). The controller script creates a directory to save forensic data collections to, and launches a DOS batch program which, in turn, kicks off each of

Forensic Programs Used	
Application/Sensor	Source
Windows Script Host v5.6	Included with MS Windows XP, SP2
logman.exe v5.1.2600.2180	Included with MS Windows XP, SP2
tshark.exe v1.2.4	Included with Wireshark v1.2.4
listdlls.exe v2.25	www.sysinternals.com
logonsession.exe v1.1	www.sysinternals.com
pslist.exe v1.28	www.sysinternals.com
tcpvcon.exe v2.54	www.sysinternals.com

Table A.1: Executable files used as part of the forensic toolkit (on the target machine). If you're unable to secure the same version, parsing the output may require rework.

the forensic tools in the toolkit in succession. For each experiment, these tools are launched by running Autoexec.vbs from the Windows console.

To use these scripts, save each of the below sections to the file names indicated. Download the executables listed in Table A and save them to the system to be monitored. Adjust file paths in batch files as necessary to match the location you saved the executables to.

A.3 Controller Script: Autoexec.vbs

```
'**********************************************************************
'* AUTHOR: Capt Joe Erskine                                           *
'* PURPOSE: Create directory to store forensic data                   *
'**********************************************************************

'**********************************************************************
'* The getDateString function builds & returns a string in the format of *
'* YYYY.MM.DD.HH.MM.SS                                                *
'**********************************************************************
Function getDateString()
Dim nowDate
nowDate = Now()
getDateString = "" & Year(nowDate) & "."
If Len(Month(nowDate)) = 1 Then
getDateString = getDateString & "0"
End If
getDateString = getDateString & month(nowDate) & "."
If Len(Day(nowDate)) = 1 Then
getDateString = getDateString & "0"
End If
getDateString = getDateString & Day(nowDate) & "."
If Len(Hour(nowDate)) = 1 Then
```

```
getDateString = getDateString & "0"
End If
getDateString = getDateString & Hour(nowDate) & "."
If Len(Minute(nowDate)) = 1 Then
getDateString = getDateString & "0"
End If
getDateString = getDateString & Minute(nowDate) & "."
If Len(Second(nowDate)) = 1 Then
getDateString = getDateString & "0"
End If
getDateString = getDateString & Second(nowDate)
End Function

'**************************************************************************
'* The makeDirectory is method is responsible for creating a directory.  *
'* The directory can be several layers deep, and if parent directories   *
'* don't exist, it creates them. If the directory already exists, it is  *
'* left alone.                                                           *
'**************************************************************************

'**************************************************************************
'* makeDirectory() function                                              *
'* Create a directory as specified by the newPath string                 *
'* adapted from...                                                       *
'* diablopup.blogspot.com/2007/04/vbscript-fun-create-file-system.html   *
'**************************************************************************
Const CONST_DIRALREADYEXISTED = -2
Const CONST_DIRCREATIONFAILURE = -1
Const CONST_DIRCREATIONSUCCESS = 0

Function makeDirectory(newPath)
```

```
Dim outputFileObject, path, count, myArray, length
Set outputFileObject = CreateObject("Scripting.FileSystemObject")
If outputfileObject.FolderExists(newPath) Then
makeDirectory = CONST_DIRALREADYEXISTED
Else
path = ""
count = 0
myArray = Split(newPath, "\")
length = UBound(myArray)
While count <= length
path = path + myArray(count) + "\"
count = count + 1
If outputfileObject.FolderExists(path) Then
Else
outputfileObject.CreateFolder(path)
End If
Wend
If outputfileObject.FolderExists(newPath) Then
makeDirectory = CONST_DIRCREATIONSUCCESS
Else
makeDirectory = CONST_DIRCREATIONFAILURE
End If
End If
End Function

'*********************************************************************
'* main() function.                                                  *
'* Create directory structure for storing forensic data, then call our   *
'* forensic tool launcher program, passing the file name             *
'*********************************************************************
Sub main()
```

```
dim forensicDataDirectory
Set sh = CreateObject("WScript.Shell")
forensicDataDirectory = "Z:\Shared Documents\SensorData\" & _
                            getDateString
Select Case makeDirectory(forensicDataDirectory)
Case CONST_DIRCREATIONSUCCESS
WScript.Echo "Created directory: " & forensicDataDirectory
sh.SendKeys "start ""Forensic Tool Launcher"" /min "
            sh.SendKeys "ForensicScan.bat """
            sh.SendKeys forensicDataDirectory & """ {ENTER}"
Case CONST_DIRCREATIONFAILURE
WScript.Echo "ERROR, could not create directory: """
            WScript.Echo forensicDataDirectory & """... Exiting script"
WScript.Quit(1)
Case CONST_DIRALREADYEXISTED
WScript.Echo "ERROR, directory: """ & forensicDataDirectory
            WScript.Echo """ already exists... Exiting script"
WScript.Quit(1)
End Select
End Sub

'**************************************************************************
'* Call main function...                                                  *
'**************************************************************************
main

'**************************************************************************
'* END OF SCRIPT                                                          *
'**************************************************************************
```

A.4 Batch program: ForensicScan.bat

```
@echo off

break=on

cls

echo.

echo *********************************************************************

echo * LFTB Live Forensic Tool Scanner v. 1.0                             *

echo * AUTHOR: Joe Erskine, Capt, USAF                                    *

echo * PURPOSE: Perform live forensic data collections                    *

echo * Log of captures being written to log file listed below.            *

echo *********************************************************************

echo.

echo Press [CTRL]+C to terminate

echo.


type C:\WINDOWS\system32\drivers\etc\services > "%~1\services.services"


cd "C:\Documents and Settings\user\Desktop\Cyber Tools\2.0 Sensors"


REM Launch separate process (command window) to capture network traffic

start "TShark" /min tshark.bat "%~1"


REM stop the threads event trace capture (if running), then start new one

logman threads stop

logman threads start


REM Launch separate process (console window) to capture process snapshots

start "ProcessSnapshot" /min processes.bat "%~1"


REM Launch Performance monitor trace using Microsoft's logman utility

REM You must configure a trace log called "threads" before running logman
```

```
logman threads start

rem All files generated here are text-based, but their filetypes
rem are customized for identifying which parser to use in our java program
rem (So, do not change file exensions if using our java app unless
rem those changes are reflected in the program as well)

REM ***********************************************************************
REM * Iteratively capture snapshots of logonsessions, tcp/udp network    *
REM * connections to processes, and dlls associated with processes       *
REM ***********************************************************************
SET i=1
:TOPOFLOOP
cls
echo.
echo %date% %time%: Initiating forensic scans (round %i%)
echo Press [CTRL]+C at any time to terminate
echo.

echo FORENSIC START TIMESTAMP: %date% %time% > %1\%i%.logonsessions
Toolkit\logonsessions.exe /p >> %1\%i%.logonsessions
echo FORENSIC STOP TIMESTAMP: %date% %time% >> %1\%i%.logonsessions

echo FORENSIC START TIMESTAMP: %date% %time% > %1\%i%.tcpvcon
Toolkit\tcpvcon.exe -a -c >> %1\%i%.tcpvcon
echo FORENSIC STOP TIMESTAMP: %date% %time% >> %1\%i%.tcpvcon

echo FORENSIC START TIMESTAMP: %date% %time% > %1\%i%.listdlls
Toolkit\listdlls.exe >> %1\%i%.listdlls
  echo FORENSIC STOP TIMESTAMP: %date% %time% >> %1\%i%.listdlls
```

```
cscript //nologo sleep.vbs 1000


SET /a i=%i%+1


GOTO TOPOFLOOP
REM ***********************************************************************
REM * END OF BATCH PROGRAM                                                *
REM ***********************************************************************
```

A.5 Batch Program: tshark.bat

```
@echo off
echo **********************************************************************
echo * Starting tshark to capture network packets                        *
echo * Press [CTRL]-C to stop capture                                    *
echo **********************************************************************
echo.
"C:\Program Files\Wireshark\tshark.exe" -p -w %1\tshark.pcap
```

A.6 Batch Program: processes.bat

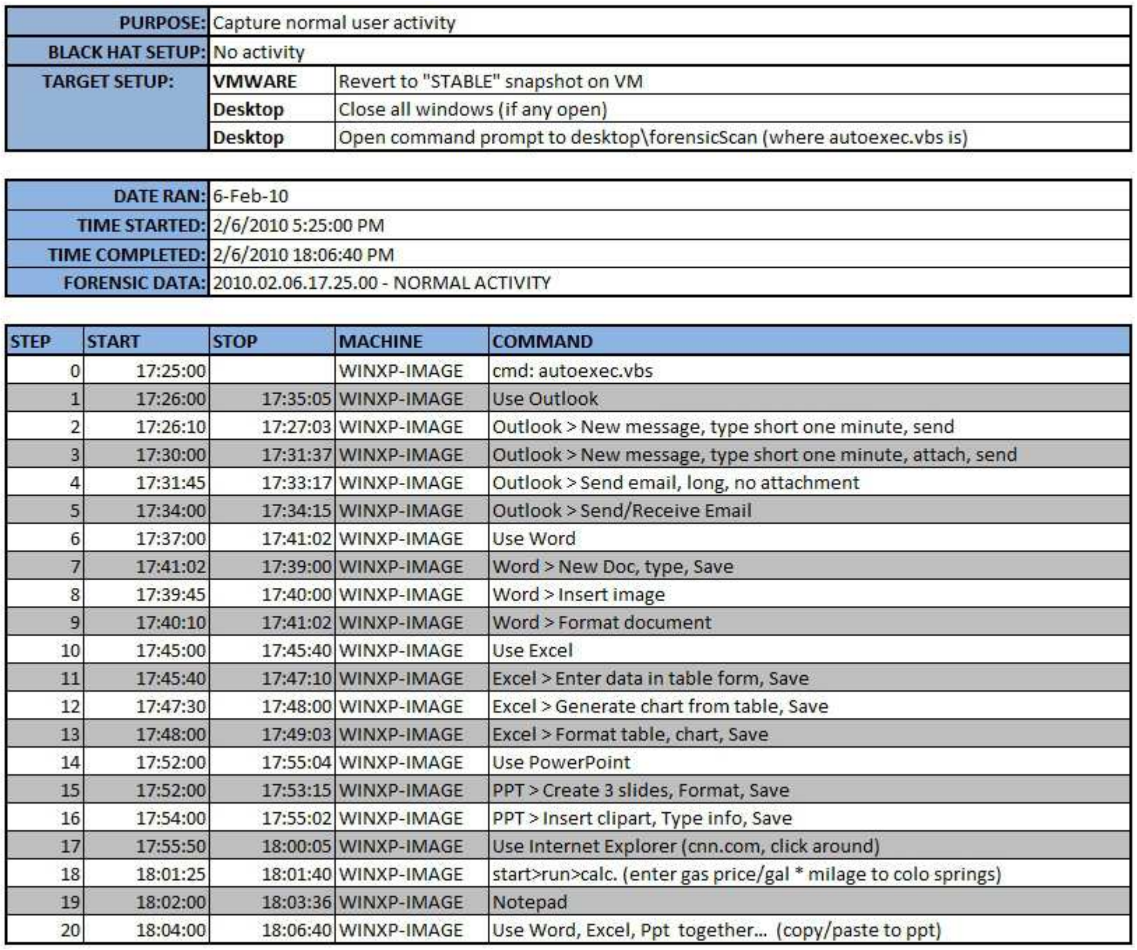
```
@echo off
echo **********************************************************************
echo * Starting process snapshots (at two-second intervals)              *
echo * Press [CTRL]-C to stop capture                                    *
echo **********************************************************************
echo.
echo Starting process snapshots to "%~1%\processes.pslistx"


sysinternals\pslist.exe -x -s -r 2 >> "%~1%\processes.pslistx"
```

Appendix B. Normal Activity Scripts

The script shown in Figure B.1 was planned in an effort to provide some notional "Normal" activity. It is not intended to capture the totality of what a person can do on a computer, but to run the computer through a few common "end user" activities, such as using a few client applications (client only and minimal client/server), using email (client/server activity), browsing web sites. In the interest of keeping collections small, the activities are admittedly more densely scheduled than a typical user may implement them. It is important to note that the purpose for these scripts is purely for data collection in support of this thesis, and not intended for user modeling research.

When running a capture, the experimenter should annotate start/stop times for tracing through the data after the collection is complete.

NOTE - Initially, the author wrote a series of WSH scripts to automate normal activity through a series of sendkeys() calls. However, while a "cool" concept, these scripts were highly dependent on system timing, and if the processor was bogged down, the scripts would fail (repeatedly). Those efforts were abandoned for this, more manual process.



PURPOSE:	Capture normal user activity	
BLACK HAT SETUP:	No activity	
TARGET SETUP:	**VMWARE**	Revert to "STABLE" snapshot on VM
	Desktop	Close all windows (if any open)
	Desktop	Open command prompt to desktop\forensicScan (where autoexec.vbs is)

DATE RAN:	6-Feb-10
TIME STARTED:	2/6/2010 5:25:00 PM
TIME COMPLETED:	2/6/2010 18:06:40 PM
FORENSIC DATA:	2010.02.06.17.25.00 - NORMAL ACTIVITY

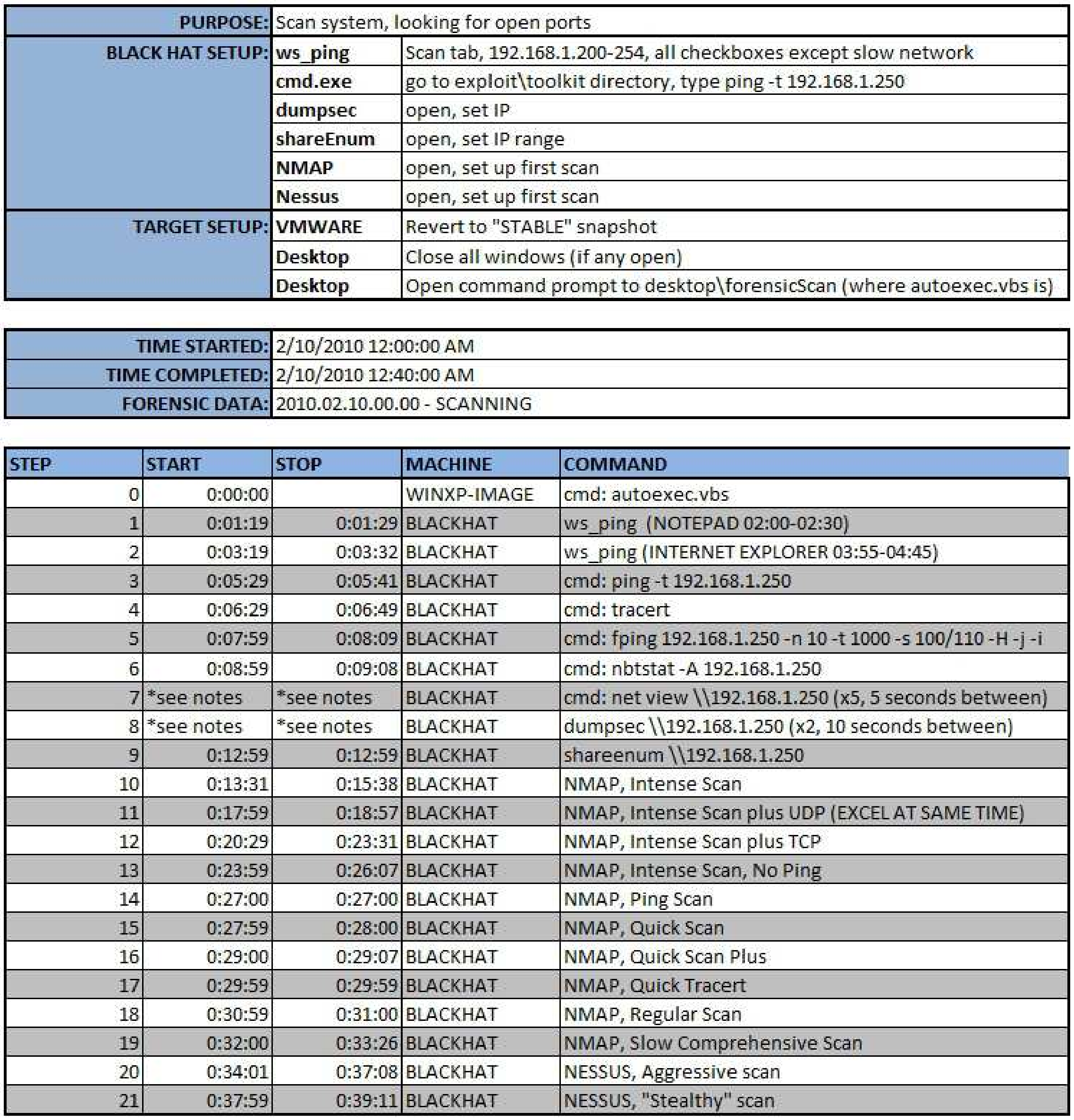
STEP	START	STOP	MACHINE	COMMAND
0	17:25:00		WINXP-IMAGE	cmd: autoexec.vbs
1	17:26:00	17:35:05	WINXP-IMAGE	Use Outlook
2	17:26:10	17:27:03	WINXP-IMAGE	Outlook > New message, type short one minute, send
3	17:30:00	17:31:37	WINXP-IMAGE	Outlook > New message, type short one minute, attach, send
4	17:31:45	17:33:17	WINXP-IMAGE	Outlook > Send email, long, no attachment
5	17:34:00	17:34:15	WINXP-IMAGE	Outlook > Send/Receive Email
6	17:37:00	17:41:02	WINXP-IMAGE	Use Word
7	17:41:02	17:39:00	WINXP-IMAGE	Word > New Doc, type, Save
8	17:39:45	17:40:00	WINXP-IMAGE	Word > Insert image
9	17:40:10	17:41:02	WINXP-IMAGE	Word > Format document
10	17:45:00	17:45:40	WINXP-IMAGE	Use Excel
11	17:45:40	17:47:10	WINXP-IMAGE	Excel > Enter data in table form, Save
12	17:47:30	17:48:00	WINXP-IMAGE	Excel > Generate chart from table, Save
13	17:48:00	17:49:03	WINXP-IMAGE	Excel > Format table, chart, Save
14	17:52:00	17:55:04	WINXP-IMAGE	Use PowerPoint
15	17:52:00	17:53:15	WINXP-IMAGE	PPT > Create 3 slides, Format, Save
16	17:54:00	17:55:02	WINXP-IMAGE	PPT > Insert clipart, Type info, Save
17	17:55:50	18:00:05	WINXP-IMAGE	Use Internet Explorer (cnn.com, click around)
18	18:01:25	18:01:40	WINXP-IMAGE	start>run>calc. (enter gas price/gal * milage to colo springs)
19	18:02:00	18:03:36	WINXP-IMAGE	Notepad
20	18:04:00	18:06:40	WINXP-IMAGE	Use Word, Excel, Ppt together... (copy/paste to ppt)

Figure B.1: "Normal activity" collections were performed by launching our sensors, followed by carrying out this series of activities. Each event's start and stop times were recorded for later analysis.

Appendix C. Cyber Attack Scripts

The script shown in Figures C.1 and C.2 were planned in an effort to provide some notional "Scanning" and "Exploit" activity. These scripts are not meant to capture the totality of what an attacker can do on a computer, but to run the computer through a few common "black hat" activities, such as using Nessus, MetaSploit and BackOrifice. In the interest of keeping collections small, the activities were split into two collection periods, and are admittedly more densely scheduled than a typical hacker who knows nothing about the system under attack may implement them. It is important to note that the purpose for these scripts is purely for data collection in support of this thesis, and not intended for user modeling research.

As with the "normal activity" capture, the experimenter should annotate start/stop times for tracing through the data after the collection is complete.

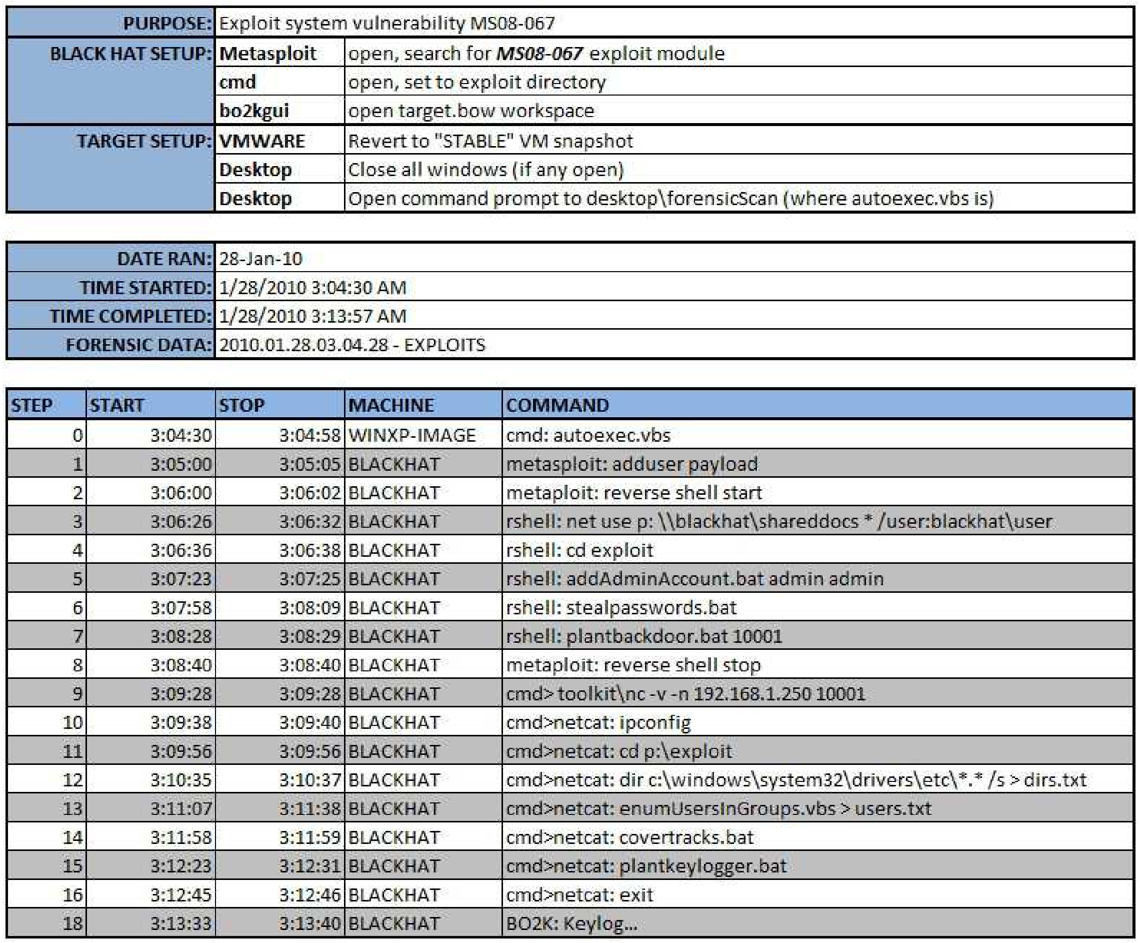


PURPOSE:	Scan system, looking for open ports	
BLACK HAT SETUP:	**ws_ping**	Scan tab, 192.168.1.200-254, all checkboxes except slow network
	cmd.exe	go to exploit\toolkit directory, type ping -t 192.168.1.250
	dumpsec	open, set IP
	shareEnum	open, set IP range
	NMAP	open, set up first scan
	Nessus	open, set up first scan
TARGET SETUP:	**VMWARE**	Revert to "STABLE" snapshot
	Desktop	Close all windows (if any open)
	Desktop	Open command prompt to desktop\forensicScan (where autoexec.vbs is)

TIME STARTED:	2/10/2010 12:00:00 AM
TIME COMPLETED:	2/10/2010 12:40:00 AM
FORENSIC DATA:	2010.02.10.00.00 - SCANNING

STEP	START	STOP	MACHINE	COMMAND
0	0:00:00		WINXP-IMAGE	cmd: autoexec.vbs
1	0:01:19	0:01:29	BLACKHAT	ws_ping (NOTEPAD 02:00-02:30)
2	0:03:19	0:03:32	BLACKHAT	ws_ping (INTERNET EXPLORER 03:55-04:45)
3	0:05:29	0:05:41	BLACKHAT	cmd: ping -t 192.168.1.250
4	0:06:29	0:06:49	BLACKHAT	cmd: tracert
5	0:07:59	0:08:09	BLACKHAT	cmd: fping 192.168.1.250 -n 10 -t 1000 -s 100/110 -H -j -i
6	0:08:59	0:09:08	BLACKHAT	cmd: nbtstat -A 192.168.1.250
7	*see notes	*see notes	BLACKHAT	cmd: net view \\192.168.1.250 (x5, 5 seconds between)
8	*see notes	*see notes	BLACKHAT	dumpsec \\192.168.1.250 (x2, 10 seconds between)
9	0:12:59	0:12:59	BLACKHAT	shareenum \\192.168.1.250
10	0:13:31	0:15:38	BLACKHAT	NMAP, Intense Scan
11	0:17:59	0:18:57	BLACKHAT	NMAP, Intense Scan plus UDP (EXCEL AT SAME TIME)
12	0:20:29	0:23:31	BLACKHAT	NMAP, Intense Scan plus TCP
13	0:23:59	0:26:07	BLACKHAT	NMAP, Intense Scan, No Ping
14	0:27:00	0:27:00	BLACKHAT	NMAP, Ping Scan
15	0:27:59	0:28:00	BLACKHAT	NMAP, Quick Scan
16	0:29:00	0:29:07	BLACKHAT	NMAP, Quick Scan Plus
17	0:29:59	0:29:59	BLACKHAT	NMAP, Quick Tracert
18	0:30:59	0:31:00	BLACKHAT	NMAP, Regular Scan
19	0:32:00	0:33:26	BLACKHAT	NMAP, Slow Comprehensive Scan
20	0:34:01	0:37:08	BLACKHAT	NESSUS, Aggressive scan
21	0:37:59	0:39:11	BLACKHAT	NESSUS, "Stealthy" scan

Figure C.1: “Scanning activity” collections were performed by launching our sensors on our target system, followed by carrying out this series of activities from the black hat system. Each event’s start and stop times were recorded for later analysis.



PURPOSE:	Exploit system vulnerability MS08-067	
BLACK HAT SETUP:	**Metasploit**	open, search for ***MS08-067*** exploit module
	cmd	open, set to exploit directory
	bo2kgui	open target.bow workspace
TARGET SETUP:	**VMWARE**	Revert to "STABLE" VM snapshot
	Desktop	Close all windows (if any open)
	Desktop	Open command prompt to desktop\forensicScan (where autoexec.vbs is)

DATE RAN:	28-Jan-10
TIME STARTED:	1/28/2010 3:04:30 AM
TIME COMPLETED:	1/28/2010 3:13:57 AM
FORENSIC DATA:	2010.01.28.03.04.28 - EXPLOITS

STEP	**START**	**STOP**	**MACHINE**	**COMMAND**
0	3:04:30	3:04:58	WINXP-IMAGE	cmd: autoexec.vbs
1	3:05:00	3:05:05	BLACKHAT	metasploit: adduser payload
2	3:06:00	3:06:02	BLACKHAT	metaploit: reverse shell start
3	3:06:26	3:06:32	BLACKHAT	rshell: net use p: \\blackhat\shareddocs * /user:blackhat\user
4	3:06:36	3:06:38	BLACKHAT	rshell: cd exploit
5	3:07:23	3:07:25	BLACKHAT	rshell: addAdminAccount.bat admin admin
6	3:07:58	3:08:09	BLACKHAT	rshell: stealpasswords.bat
7	3:08:28	3:08:29	BLACKHAT	rshell: plantbackdoor.bat 10001
8	3:08:40	3:08:40	BLACKHAT	metaploit: reverse shell stop
9	3:09:28	3:09:28	BLACKHAT	cmd> toolkit\nc -v -n 192.168.1.250 10001
10	3:09:38	3:09:40	BLACKHAT	cmd>netcat: ipconfig
11	3:09:56	3:09:56	BLACKHAT	cmd>netcat: cd p:\exploit
12	3:10:35	3:10:37	BLACKHAT	cmd>netcat: dir c:\windows\system32\drivers\etc\*.* /s > dirs.txt
13	3:11:07	3:11:38	BLACKHAT	cmd>netcat: enumUsersInGroups.vbs > users.txt
14	3:11:58	3:11:59	BLACKHAT	cmd>netcat: covertracks.bat
15	3:12:23	3:12:31	BLACKHAT	cmd>netcat: plantkeylogger.bat
16	3:12:45	3:12:46	BLACKHAT	cmd>netcat: exit
18	3:13:33	3:13:40	BLACKHAT	BO2K: Keylog...

Figure C.2: "Exploit activity" collections were performed by launching our sensors on our target system, followed by carrying out this series of activities from the black hat system. Each event's start and stop times were recorded for later analysis.

Bibliography

1. URL `http://www.symantec.com/norton/antivirus`.
2. "AFPC Demographic Report Builder Website". URL `http://wwa.afpc.randolph.af.mil/demographics/`.
3. "jNetPcap API Library". URL `http://www.jnetpcap.com, year=2009`.
4. "Wireshark Website". URL `http://www.wireshark.org`.
5. "Joint Publication 1-02, Department of Defense Dictionary of Military and Associated Terms", April 2001 (As ammended through 19 Aug 2009).
6. "Joint Publication 5-0, Joint Operational Planning", April 2001 (As ammended through 19 Aug 2009).
7. "Joint Publication 6-0, Joint Communication Systems", Mar 2006.
8. "Insecure.org Website", 2010. URL `http://insecure.org`.
9. Akkan, H. "DIGITAL FORENSICS".
10. ALEXANDER, S. "finding malware on compromised Windows machines".
11. Axelsson, S. "Intrusion detection systems: A survey and taxonomy". *Depart. of Computer Engineering, Chalmers University, Tech. Rep*, 99–15, 2000.
12. Baader, F., A. Bauer, P. Baumgartner, A. Cregan, A. Gabaldon, K. Ji, K. Lee, D. Rajaratnam, and R. Schwitter. *A Novel Architecture for Situation Awareness Systems*, 77. 2009.
13. Bace, R. and P. Mell. "NIST special publication on intrusion detection systems". *SP800-31, NIST, Gaithersburg, MD*, 2001.
14. Barham, P., R. Isaacs, R. Mortier, and D. Narayanan. "Magpie: real-time modelling and performance-aware systems". *9th Workshop on Hot Topics in Operating Systems, Lihue, Hawaii.* 2003.
15. Bass, T. "Intrusion detection systems and multisensor data fusion". 2000.
16. Bernaille, L., R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian. "Traffic classification on the fly". *ACM SIGCOMM Computer Communication Review*, 36(2):26, 2006.
17. Black, N. "A Comparative Analysis of Two Approaches of Computer Network Intrusion Detection". *cognita*, 25.
18. Brugger, S.T. "KDD Cup99 dataset (Network Intrusion) considered harmful". *KDnuggets newsletter*, 7(18):15, 2007.

19. Brugger, S.T. and J. Chow. "An assessment of the DARPA IDS Evaluation Dataset using Snort". *UCDAVIS department of Computer Science*, 2007–1, 2007.

20. Carvey, H. "Malware analysis for windows administrators". *Digital Investigation*, 2:19e22, 2005.

21. Chapman, P., J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, and R. Wirth. "CRISP-DM 1.0: Step-by-step data mining guide". *SPSS inc*, 78, 2000.

22. Chari, S.N. and P.C. Cheng. "Bluebox: A policy-driven, host-based intrusion detection system". *ACM Transactions on Information and System Security (TISSEC)*, 6(2):173–200, 2003.

23. CHOO, V. and L. Scheiderich. "Information Operations Innovation Network (IOIN) Demonstration". 2006.

24. CJCS, Chairman U.S. Joint Chiefs of Staff. *The National Military Strategy for Cyberspace Operations*. Department of Defense, Washington, DC, December 2006.

25. Clausewitz, Carl von. *On War*. Routledge & Kegan Paul, 1968.

26. CNN. "Thrift store MP3 player contains secret military files", 2009. URL `http://www.cnn.com/2009/TECH/01/27/confidential.mp3.player/index.html`.

27. Darren, P. "Number of viruses to top 1 million by 2009", 2008. URL `http://www.computerworld.com/s/article/9075118/`.

28. Debar, H., M. Dacier, and A. Wespi. "Towards a taxonomy of intrusion-detection systems". *Comput. Networks*, 31(8):805–822, 1999.

29. Demuth, H. and Hagan M. Beale, M. "The Neural Network Toolbox 6 for MATLAB User's Guide". 2009.

30. Denning, D.E. "An intrusion-detection model". *IEEE Transactions on software engineering*, 13(2):222–232, 1987.

31. Dey, A.K. "Understanding and using context". *Personal and ubiquitous computing*, 5(1):4–7, 2001.

32. Endsley, M.R. "Design and evaluation for situation awareness enhancement". *Human Factors and Ergonomics Society Annual Meeting Proceedings*, volume 32, 97–101. Human Factors and Ergonomics Society, 1988.

33. Endsley, M.R. "Theoretical underpinnings of situation awareness: A critical review". *Situation awareness analysis and measurement*, 3–32, 2000.

34. Florez, G. "Analyzing system call sequences with Adaboost". *Proceedings of the 2002 International Conference on Artificial Intelligence and Applications (AIA), Malaga, Spain.* 2002.

35. Fyfe, Colin. "Lecture Notes in Computer Science: Artificial Neural Networks and Information Theory", 2000.

36. Gonzalez, J.A. "Numerical Analysis for Relevant Features in Intrusion Detection (NARFid)", 2009.

37. Gregg, M. *Certified ethical hacker*. Que Certification, [Indianapolis, Ind.], 2006.

38. Haag, C.R., G.B. Lamont, P.D. Williams, and G.L. Peterson. "An artificial immune system-inspired multiobjective evolutionary algorithm with application to the detection of distributed computer network intrusions", 2007.

39. Hall, D.L. and J. Llinas. "An introduction to multisensor data fusion". *Proceedings of the IEEE*, 85(1):6–23, 1997.

40. Haykin, S. "Neural networks and learning machines". *Prentice-Hall*, 2008.

41. Howard, M., D. LeBlanc, and J. Viega. *19 deadly sins of software security*. McGraw-Hill/Osborne, 2005.

42. Ilgun, K., R.A. Kernmeter, and P.A. Porras. "State transition analysis: A rule-based intrusion detection approach". *IEEE transactions on software engineering*, 1995.

43. Kohavi, R. and G.H. John. "Wrappers for feature subset selection". *Artificial intelligence*, 97(1-2):273–324, 1997.

44. Laboratory, Lawrence Berkeley National. 2009. URL `http://bro-ids.org`.

45. Lamport, L. "Time, clocks, and the ordering of events in a distributed system". 1978.

46. Lazarevic, A., L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava. "A comparative study of anomaly detection schemes in network intrusion detection". *Proceedings of the Third SIAM International Conference on Data Mining*. 2003.

47. Lee, J.H., J.H. Lee, S.G. Sohn, J.H. Ryu, and T.M. Chung. "Effective Value of Decision Tree with KDD 99 Intrusion Detection Datasets for Intrusion Detection System". *Advanced Communication Technology*, 2:1170–1175, 2008.

48. Lee, W. and S.J. Stolfo. "A framework for constructing features and models for intrusion detection systems". *ACM Transactions on Information and System Security (TISSEC)*, 3(4):227–261, 2000.

49. Lee, W., S.J. Stolfo, P.K. Chan, E. Eskin, W. Fan, M. Miller, S. Hershkop, and J. Zhang. "Real time data mining-based intrusion detection". *Proceedings of DISCEX II*, 89–100. Citeseer, 2001.

50. Liebrock, L., N.M. Socorro, and C. Veitch. "WINDOWS DIGITAL FORENSICS TOOLKIT: An Analysis of Digital Forensics Tools".

51. Liu, H. and L. Yu. "Toward integrating feature selection algorithms for classification and clustering". *IEEE Transactions on knowledge and data engineering*, 491–502, 2005.

52. Makanju, A.A.O., A.N. Zincir-Heywood, and E.E. Milios. "Clustering event logs using iterative partitioning". *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 1255–1264. ACM New York, NY, USA, 2009.

53. McClure, S., J. Scambray, and G. Kurtz. *Hacking Exposed 6: Network Security Secrets & Solutions*. McGraw-Hill Osborne Media, 2009.

54. McHugh, J. "Testing intrusion detection systems: A critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory". *ACM Transactions on Information and System Security*, 3(4):262–294, 2000.

55. Nguyen, N., P. Reiher, and G.H. Kuenning. "Detecting insider threats by monitoring system call activity". *Proc. of IEEE Workshop on Information Assurance*, volume 38. Citeseer, 2001.

56. Noguchi, Yuki. "Lost a BlackBerry? Data Could Open A Security Breach". *Washington Post*, 2005. URL `http://www.washingtonpost.com/wp-dyn/content/article/2005/07/24/AR2005072401135.html`.

57. Okolica, J., J.T. McDonald, G.L. Peterson, R.F. Mills, and M.W. Haas. "Developing Systems for Cyber Situational Awareness". 46, 2009.

58. Russell, S. and P. Norvig. "Artificial intelligence: a modern approach". *New Jersey*, 1995.

59. Russinovich, Mark. "Microsoft Sysinternals Suite". URL `http://technet.microsoft.com/en-us/sysinternals/bb842062.aspx`.

60. Russinovich, M.E. and D.A. Solomon. *Microsoft Windows Internals, Microsoft Windows Server 2003, Windows XP, and Windows 2000*. Microsoft Press, 2005.

61. Schilit, B., N. Adams, R. Want, et al. "Context-aware computing applications". *Proceedings of the workshop on mobile computing systems and applications*, 85–90. Citeseer, 1994.

62. Shilland, G.R. and U. Major. "Host-Based Multivariate Statistical Computer Operating Process Anomaly Intrusion Detection System (PAIDS)", 2009.

63. Simache, C., M. Kaâniche, and A. Saidane. "Event log based dependability analysis of Windows NT and 2K systems". *Proc. of the 2002 Pacific Rim International Symposium on Dependable Computing (PRDC02)*. 2002.

64. Sourcefire, 2009. URL `http://www.snort.org`.

65. Steinberg, A.N., F.E. White, and C.L. Bowman. "Revisions to the JDL data fusion model". Environmental Research Institute of Michigan, Arlington VA, 1999.

66. Stotz, A. and M. Sudit. "INformation Fusion Engine for Real-time Decision-making (INFERD): a perceptual system for cyber attack tracking". *Proceedings of the 10th IEEE International Conference on Information Fusion*, 1–8. 2007.

67. Sullivan, S. "News from the Lab: Weblog Q & A". URL `http://www. f-secure. com/weblog/archives/00001198.html, 2007.`

68. Waits, C., J.A. Akinyele, R. Nolan, and L. Rogers. "Computer Forensics: Results of Live Response Inquiry vs. Memory Image Analysis". 2008.

69. Wang, W., X. Zhang, and S. Gombault. "Constructing attribute weights from computer audit data for effective intrusion detection". *The Journal of Systems & Software*, 2009.

70. Wu, X., V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G.J. McLachlan, A. Ng, B. Liu, P.S. Yu, et al. "Top 10 algorithms in data mining". *Knowledge and Information Systems*, 14(1):1–37, 2008.

71. Zaraska, K. "Prelude IDS: current state and development perspectives". *URL http://www. prelude-ids. org/download/misc/pingwinaria/2003/paper. pdf*, 2003.

Vita

Captain Joseph R. Erskine graduated from Cabrillo High School in Lompoc, California in 1991. He entered undergraduate studies at Texas Lutheran University, Seguin, Texas and in 2002, he obtained a Bachelor of Science degree, cum laude, in Computer Science. He was commissioned in the United States Air Force in June of 2004. He is married with two children.

Captain Erskine entered the United States Air Force in 1992 as a Communications-Computer Systems Programmer. He was assigned to the San Antonio Computer Services Center as a Small Computer Analyst in San Antonio, Texas, where he was the 1993 Defense Information Systems Agency (DISA) nominee to Top 12 Outstanding Airmen of the Year. In 1994, he was assigned to the Air Force Personnel Center (AFPC), Randolph Air Force Base, Texas, first as a Network Systems Administrator, then as an Internet Applications Programmer, where he authored the prototype "Virtual Military Personnel Flight" (vMPF) application. He has proudly served two tours as a volunteer Ceremonial Guardsman with the Randolph Air Force Base Honor Guard. Following graduation from Texas Lutheran University, he was accepted to the United States Air Force Officer Training School. Upon his commissioning in 2004, he was assigned as Crew Commander, then as the Commander, Systems Integration Flight, to the Pacific Network Operations and Security Center (PACAF NOSC), Hickam Air Force Base, Hawaii. In 2006, he was assigned as the Deputy Commander, Command and Control Systems Flight, to the 56th Air Communication Squadron (ACOMS), supporting the Pacific Air Operations Center. In 2008, he was selected to attend the Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio. Upon graduation, Captain Erskine will be assigned as a Computer Science Instructor to the United States Air Force Academy, Colorado Springs, Colorado.

Permanent address: 2950 Hobson Way
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)* 25–03–2010	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From — To)* Sept 2008 — Mar 2010

4. TITLE AND SUBTITLE Developing Cyberspace Data Understanding: Using CRISP-DM for Host-based IDS Feature Mining	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Joseph R. Erskine, Capt, USAF	**5d. PROJECT NUMBER** 10-311
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology, Graduate School of Engineering and Management (AFIT/EN), 2950 Hobson Way, WPAFB OH 45433-7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GCS/ENG/10-01
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Dr. Robert L. Herklotz, Air Force Office of Scientific Research, AFMC, 801 North Randolph Street, Rm 732, Arlington VA 22203-1977, 703-696-9544 (DSN: 426) robert.herklotz@afosr.af.mil	**10. SPONSOR/MONITOR'S ACRONYM(S)** AFOSR/NL
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approval for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Current intrusion detection systems generate a large number of specific alerts, but do not provide actionable information. Many times, these alerts must be analyzed by a network defender, a time consuming and tedious task which can occur hours or days after an attack occurs. Improved understanding of the cyberspace domain can lead to great advancements in Cyberspace situational awareness research and development. This thesis applies the Cross Industry Standard Process for Data Mining (CRISP-DM) to develop an understanding about a host system under attack. Data is generated by launching scans and exploits at a machine outfitted with a set of host-based data collectors. Through knowledge discovery, features are identified within the data collected which can be used to enhance host-based intrusion detection. By discovering relationships between the data collected and the events, human understanding of the activity is shown. This method of searching for hidden relationships between sensors greatly enhances understanding of new attacks and vulnerabilities, bolstering our ability to defend the cyberspace domain.

15. SUBJECT TERMS

situational awareness, intrusion detection, data mining, threat modeling

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Dr. Gilbert Peterson
U	U	U	UU	136	**19b. TELEPHONE NUMBER** *(include area code)* (937)255–3636, x4281; gilbert.petersonafit.edu

Standard Form 298 (Rev. 8–98)
Prescribed by ANSI Std. Z39.18